JYVÄSKYLÄ STUDIES IN COMPUTING
72

Dmytro Zhovtobryukh

Context-aware Web Service Composition

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 72

Dmytro Zhovtobryukh

Context-aware Web Service Composition

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksessa (Ag Aud. 2)
joulukuun 28. päivänä 2006 kello 13.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in the Building Agora, Ag Aud. 2, on December 28, 2006 at 1 p.m.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2006

Context-aware
Web Service Composition

JYVÄSKYLÄ STUDIES IN COMPUTING 72

Dmytro Zhovtobryukh

Context-aware Web Service Composition

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2006

Editors
Tommi Kärkkäinen
Department of Mathematical Information Technology, University of Jyväskylä
Irene Ylönen, Marja-Leena Tynkkynen
Publishing Unit, University Library of Jyväskylä

URN:ISBN:951392727X
ISBN 951-39-2727-X (PDF)

ISBN 951-39-2723-7 (nid.)
ISSN 1456-5390

Copyright © 2006, by University of Jyväskylä

Jyväskylä University Printing House, Jyväskylä 2006

ABSTRACT

Zhovtobryukh, Dmytro
Context-aware Web Service Composition
Jyväskylä; University of Jyväskylä, 2006, 290p.
(Jyväskylä Studies in Computing
ISSN 1456-5390; 72)
ISBN 951-39-2727-X (PDF), 951-39-2723-7 (nid.)
Finnish summary
Diss.

This doctoral dissertation is aimed at the creation of comprehensive and innovative solution of the Web Service Composition problem. More specifically, the thesis focuses on the methodological aspect of the solution sought, trying to build up a universal type of service composition mechanism, the applicability of which would not be limited to a bundle of middleware and data representation standards. The solution relies on the principles of openness and technological neutrality and is based on rigorous analysis of existing practical and theoretical contributions, efficiently reusing them.

The work presented within this thesis is positioned at the rich intersection of Service Oriented Computing, Semantic Web and Context-Aware Computing paradigms. Among the goals there are serious challenges, such as automated service composition, goal-driven service composition, and proactive service composition. Although these challenges have already been met by some prominent methodological solutions, this dissertation work addresses them once more, partially by revising the existing solution approaches and partially by utilizing its own innovative approach.

The innovativeness of the proposed composition approach becomes evident with the introduction of the concept of Context-Aware Composite Web Service. Context is considered as a new facet of Web Services' description and a new dimension of Web Service composition control. Incorporating context awareness into Web Services and Web Service composition frameworks not only increases quality, usability and flexibility of composite services, but also enhances the functionality of core composition mechanisms by improving relevancy, robustness and scalability of produced compositions.

The design methodology is based on the mature formal modeling tool – Petri nets. This formalism possesses some outstanding qualities for a service composition modeling tool.

Keywords: Web Service, service composition, Semantic Web service, context, context awareness, automation, reconfigurability, Petri nets.

Author

Dmytro Zhovtobryukh
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Supervisors

Professor Pekka Neittaanmäki
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Professor Veikko Hara
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Professor Vagan Terziyan
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Reviewers

Professor Gennady Yanovsky
Telecommunications Networks Department
St. Petersburg State University of Telecommunications
Russian Federation

Professor Vadim A. Ermolayev
Department of Mathematical Modeling and Information Technology
Zaporozhye National University
Ukraine

Opponent

Professor Olli Martikainen
Department of Computer Science
University of Oulu
Finland

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Professor Pekka Neittaanmäki for his valuable all-round support of my doctoral research work during all these years. This thesis would simply not have materialized without him.

I am especially thankful to Professor Vagan Terziyan for his wise and masterly guidance and high-quality technical assistance. Though he joined our team at a quite late stage of my doctoral research, I doubt I would have made it this far without his invaluable and generous investment into my research.

I also thank Dr. Veikko Hara for encouraging me with interesting practical ideas and providing me with an industrial view on my research.

I also am grateful to COMAS Graduate School and Department of Mathematical Information Technology of the University of Jyväskylä, who have been providing me with financial support during the period of my doctoral research. A special thank goes to the Research and Training Foundation of TeliaSonera, who awarded me some extra funding in 2004, and to Ella and Georg Ehrnrooth Foundation and Ellen and Artturi Nyssönen Foundation, who commited the same contribution to my work in 2005.

Part of this research work has been carried out in conjunction with InBCT (Innovations in Business, Communications and Technology) research project, held at Agora Center and supported by TEKES in 2002-2003.

I want to thank Professor Gennady Yanovsky and Professor Vadim A. Ermolayev for reviewing the manuscript and providing me with valuable comments. I wish to thank Steve Legrand for correcting the language of the manuscript. I am also thankful to Mikko Vapa for his generous help in writing the Finnish summary for my thesis.

It was nice to work alongside with my colleague Nataliya Kohvakko. I am thankful to her for her invigorating support whenever I have been close to loosing my motivation.

I am particularly grateful to my beloved wife Anna for her patience during this challenging and sometimes difficult period in our lives, and for her unwavering belief in my capability and success. My hearty thanks to my little daughter Alexandra for providing an additional motivation factor for my work. I am happy that you exist in my life.

Finally, my heartfelt thanks to my parents, who taught me what the most important things in life are. Without the education they gave me I hardly ever picked this particular path in my life.

Jyväskylä
15th December 2006
Dmytro Zhovtobryukh

ABBREVIATIONS

AI	Artificial Intelligence
API	Application Program Interface
BPEL4WS	Business Process Execution Language for Web Services
BPML	Business Process Modeling Language
CoI	Context of Interest
C-OWL	Context Web Ontology Language
DAML	DARPA Agent Markup Language
DAML-S	DAML for Services
ebXML	electronic business eXtensible Markup Language
EO	Enterprise Ontology
ESOA	Extended Service Oriented Architecture
FTP	File Transfer Protocol
GDL4WSAC	Goal Description Language for Web Service Automatic Composition
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HTN	Hierarchical Task Network
HTTP	Hyper-Text Transfer Protocol
IP	Internet Protocol
OASIS	Organization for the Advancement of Structured Information Standards
OS	Operation System
OSI	Open Systems Interconnection
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
PC	Personal Computer
QoS	Quality of Service
RDF	Resource Description Framework
RDF-S	Resource Description Framework Schema
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
SPF	Service Portability Framework
SRM	Service Reference Model
SW	Semantic Web
SWS	Semantic Web Service
UDDI	Universal Description and Discovery Interface
URI	Universal Resource Identifier

W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
WS	Web Service
WSA	Web Services Architecture
WS-BPEL	Web Service Business Process Execution Language
WS-CDL	Web Service Choreography Description Language
WSCI	Web Service Choreograhy Interface
WSDL	Web Service Description Language
WSFL	Web Services Flow Language
WSMF	Web Service Modeling Framework
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
XML	eXtensible Markup Language
XSD	XML Schema Definition

LIST OF FIGURES

CONTENTS

1 INTRODUCTION AND MOTIVATION

In this opening Chapter of the thesis we observe contemporary trends in the modern distributed computing and present an outlook for the future evolution of it. We discuss some of the most influential ideas and valuable insights in the domain of mobile and pervasive computing as the inspiration for development of our own ideas, which are presented further in this thesis. In this Chapter we also present a compressed view of our previous work on the Service Portability Framework, which consequently motivated and laid the foundation for the major contributions we committed in this dissertation.

1.1 Modern Computing Perspective

The era of computer communications started in the mid-1970s with emergence the distributed computing paradigm [29]. Its initial objective was to build up remote communication and remote information access through a set of locally interconnected personal computers. But with development of computing and networking technology, the paradigm expanded its scope to cover various types of transparencies, scalability and availability concerns, and security and reliability issues. The success of distributed computing and communications resulted in the emergence of the Internet, a computer network system that has managed to link millions of stationary desktop computers worldwide.

Mobile computing [97] appeared to be a logical extension of the distributed systems view. In the early 1990s, with the design of pioneering portable computers, it became evident that the old paradigm required serious revision if it was to meet the requirements of the future ahead. Substantial enhancements had to be made initially to address issues related to mobility. Although the first portable computers were not for mobile use, the time of full-fledged mobile devices was not far off. To facilitate their use, a shift to mobile networking and mobile information access was necessary.

Being a natural offshoot of distributed computing, mobile computing is rapidly developing at the present time. It has managed to bring a new level of transparency to computing and communications while setting even higher usability standards. Although it has not reached its apex in its present phase yet, mobile computing is already a subject of the next wave of evolution. The coming decade is predicted to be the period of a smooth transition to a new computing paradigm called pervasive computing [96, 98]. As the descendant of mobile computing, pervasive computing unswervingly follows its design and operational principles and unobtrusively extends them to further improve the levels of transparency and usability. Pervasive computing is currently in the focus of a number of successful research and development projects, such as Aura project at Carnegie Mellon University [45], MIT project Oxygen, Portolano project at the University of Washington [37], Endeavour project at the University of California at Berkley, Sentient Computing of AT&T Laboratories, EasyLiving project of Microsoft's Research Vision Group, and CoolTown initiative of Hewlett-Packard.

While today's telecommunications world features a number of communication technologies exhibiting remarkable technological diversity, these current technologies are often, to a certain extent, functionally narrow. Moreover, since most of them are approaching their maturity, it is extremely difficult, if not impossible, to either introduce significant technological enhancements to them or to replace them with other technologies.

Modern communication standards can be roughly split into fixed and mobile technologies. The fixed sector is much more mature and well developed, and is, to a large extent, represented by the Internet, the nearly global computer network system. The Internet relies on interconnected packet-switched wired networks, which allow the provisioning of rich data and multimedia services at high speeds. This, along with the robust network technology and global accessibility, makes the Internet an attractive communication standard. On the opposite side are the mobile communication technologies that primarily utilize circuit-switched, voice-oriented wireless networks and, most importantly, establish advanced support for user mobility, which is missing in the Internet. However, wireless technologies constantly suffer from low data rates, and therefore lack support for data and multimedia. Wireless LANs stand slightly apart from the rest of the mobile technologies. They provide a reasonable compromise between mobile systems'

flexibility and the technical strength of the Internet. However, they suffer from certain drawbacks such as security concerns.

The Internet could be a perfect foundation for the next generation communications because of its global coverage, robust network vehicle and high speed. Furthermore, it has been the base of the information society for years, and the majority of services required for its role have already been developed within it. These services require mere adjustments to meet the demands of tomorrow. Unfortunately, the fixed nature of the Internet makes it ill-suited for mobility support and ubiquitous access. In addition to highlighting the fundamental challenges inherent in distributed systems, mobile computing places new requirements on communication systems in general. Frequent handoffs and disconnected operations force the system to become ever more dynamic and adaptive. Ubiquitous access creates further challenges to the terminal equipment and to the network system, which need to handle dynamic, integrated and personalized access to the available services.

With the gradual growth of diversity in communications, the tendency to make contemporary standards converge becomes increasingly vital. Modern communications apparently are starting to move towards a global, integrated computing environment. The idea of such a convergence is to substitute the impressive variety of today's communication systems with a single communication environment that would preserve the most significant contributions of its predecessors. There are two major approaches to achieve this goal.

The revolutionary approach consists of the design of a totally new technological standard that is effectively compliant with the above demands. It may imply a purely novel technological basis, software and hardware architectures, although the best achievements of the modern technologies could be reused. Not surprisingly, the most comprehensive and promising field for this approach is found within the pervasive computing paradigm.

The alternative approach is rather extensive and can be considered evolutionary. It focuses on the integration of the present communication standards into a single system. This concept currently seems more reasonable since it does not require a complete reconstruction of the existing systems. Instead, it entails the technological reinforcement of today's communication platforms in order to create an integrated global communication environment. Although it appears quite complicated, it would make contemporary communication systems interoperable. Terminals, network architectures, services and even users may be highly diverse when considering systems of various kinds. These systems must be significantly modified to introduce a unified system platform that empowers their interoperability. Whether such integration is tight or loose depends on many factors, such as how flexible the system is as far as modifications are concerned, and how affordable these modifications are for the current network operators.

1.2 Ultimate Perspective: Pervasive Computing

The concept of pervasive computing, also called ubiquitous computing, was initially proposed by Mark Weiser in the early 1990s [113, 114] and was established to denote a new paradigm of distributed computing. Being a purely user-centric paradigm, pervasive computing focuses on the demands of a user surrounded by a computing environment. The environment is no longer seen as a separate computing facility; rather it is embedded into the physical environment so that the user lives in it naturally. A pervasive computing environment is saturated with computing and communication capabilities, yet gracefully integrated with human users [98]. We will simply use word "environment" later in this section to refer to the combination of pervasive computing environment and physical surroundings in which users are involved.

The concept of invisibility arises from this point. As a matter of fact, truly appropriate and advanced technology must weave itself into the background, allowing the user not to focus on it but merely concentrate on the task (s)he is performing using this technology. Conversely, modern personal computer is brought to the foreground and, as a consequence, the user not only makes a conscious decision to use it, (s)he must learn in advance how to operate it. One good example of invisible technology (which is familiar to almost everyone) is utilization of standard setting for food intake, e.g. fork and knife for an average European. Even though we must pay focused attention to these devices when we pick them up, we do not generally notice them while using them during eating. Instead, we mostly concentrate on tasting and/or other activities some of which can be irrelevant to eating. Thus, our general observation is that invisible technology is not purely invisible in the literal sense of the word. It can be seen or otherwise detected by the user, but the most important thing is that it does not require to be seen in order to be utilized.

Another important aspect of pervasive computing is immersion. Since the computing environment is embedded in the physical neighborhood and is effectively invisible to the user, one becomes immersed in a new type of reality that combines the physical surroundings with the computing system. In contrast to virtual reality environments, the real world in a pervasive system is not replaced by an imaginary one but is just altered appropriately. Invisibility and immersion are the properties that make computing really natural and user-friendly.

Smart spaces provide the way to build up invisibility and immersion. By embedding computing and communication facilities into physical surroundings, they facilitate the fusion of the computing and physical worlds. This enables computing technology to disappear in the physical neighborhood and brings ubiquity into communications. Currently, smart spaces are unable to provide a wide coverage; they penetrate only small areas, such as rooms or limited outdoor

spaces. The "smartness" of smart spaces lies in their capability to track physical surroundings and perhaps control some of their characteristics. The most common example of such technology is an environmental control facility in a hotel room, which is capable of adjusting the temperature and humidity inside the room or even turning lights on after detecting a person inside.

Regardless of the capabilities of small-sized smart spaces, a pervasive computing environment highly populated with computing devices has to embrace the entire physical surroundings. For this reason, communication facilities available throughout the environment must be universal. The user should be able to obtain access to computing facilities practically anywhere. To achieve such ubiquity, computing devices within the environment should be networked through several types of media, perhaps both wireless and wired, with disparate communication conditions. For instance, short-range wireless connectivity can be used to link devices within the same space, medium-range wireless for connections inside the building, and high-speed fixed connections to exchange data between distant locations.

Computing devices in pervasive environments represent conventional tools for acquiring access to and manipulation of the services that are available within the environment. Those devices are as simple as possible to master and to use. They are envisioned to replace such standard communication tools as pens, paper, work desks, cord phones, etc., and to automatically store, process, and manage information via specific applications. However, these devices also have to possess distinct sets of functionalities. A straightforward classification of pervasive terminal equipment comprises three major groups of devices designated with respect to the functionality seen in today's computing devices. For illustration, the Xerox PARC's implementation offers three types of devices. One of these is a tiny wireless device called the Tab [113]. Its functions resemble those of mobile phones and pocket PCs we currently have. Another group of devices, called Pads, are designed to substitute modern laptop and desktop computers. Pads may have both wireless and wired connectivity, which brings them closer to portable computers. The third type of pervasive computing device is the Liveboard. Liveboards are large fixed devices enabling cooperative work. They are envisioned to replace whiteboards in meeting rooms, extending their functionality noticeably.

Applications in pervasive computing should be seen not as an interface to users but as a specific functionality for accomplishing service-related tasks, similarly to personalized user agents [17]. Personalization is a subject of great importance since, in the pervasive paradigm, while terminals are not necessarily personalized for every single user, applications most likely are. The applications even be able to migrate between terminals, tracking the user's movements and performing tasks on the move. Such capability helps enabling personal user mobility and is known as Teleporting or the "follow-me" principle [13]. Finally, the applications are adaptive. They are capable of self-adjusting, e.g. they adjust their

structure or execution flow, according to the changes occurring through user's demands, communication conditions and computing environment. The majority of the tasks can be automated in pervasive computing. Due to adaptability, applications can autonomously manage personal information, schedule appointments, find other users within the environment, and so on.

Context-awareness (see Chapter 2 for a rigorous definition and detailed description of the concept) is a capability of the applications to take into account various environmental data and any change in that data for a consequent adaptation of the program's behavior. Such data is widely referred to as context [41]. Various applications may be interested in diverse contexts and treat them differently when pursuing their specific goals. They might need a standardized context service, possibly based on Semantic Web ontologies [24, 35] (see Chapter 2 for the discussion of Semantic Web and ontologies), to utilize the context in a really useful and efficient manner. To facilitate efficient context utilization, a pervasive computing environment should provide efficient means for applications to acquire context. Fortunately, smart spaces present a solid low-level foundation for the development of robust environment monitoring and context sensing techniques. "Wearable computing" can give context-awareness additional perspectives in directions such as location sensitivity and, especially, in capturing user state and intent.

Pervasive computing addresses an interesting contemporary viewpoint of distributed computing. It represents a more natural computing paradigm from the viewpoint of user. In addition to that, pervasive computing resolves, in a graceful and efficient manner, some crucial challenges of modern computing [97], but it also poses some new challenges [30, 96]. Although pervasive environments are not yet widely deployed, some of the solutions proposed within this approach can be effectively adopted now to meet related challenges in current communication environments, challenges such as ubiquitous access, mobility, and adaptive applications. These are non-trivial and important issues for network interoperability problem as well. Enhancing them to a new qualitative level, pervasive computing identifies a long-term goal of network integration. As these issues will gradually be solved in the context of network integration, pervasive computing will endure on its path to the prospering future.

1.3 Challenges

Current services are designed to operate in specific communication environments. The approach is rather awkward as far as bringing ubiquity into computing and communication to prospective customers is concerned, especially taking into account the growing tendency to integrate modern communication systems. But

regardless of whether two adjacent network systems are interconnected or not, the operating service application in the majority of cases should be immediately terminated and reinitiated as the user's terminal crosses the border between the two systems and reassigns its connection to the new one in its range. This requirement can cause a problem due to the lack of appropriate infrastructure support enabling seamless operation of service application throughout multiple network systems. However, even where it exists, such infrastructure support does not spread on more than two systems. Needless to say that beside apparent performance issues this, beyond doubt, distracts potential users. Therefore it is highly desirable to provide customers with services that roam across interconnected network systems and even various devices, in an effectively transparent and continuous manner. To achieve that, pervasive computing environments, richly endowed with ambient intelligence, would be of great assistance. Unfortunately, they are still far from being widespread. Nonetheless, certain steps towards more universal service provisioning and seamless service consumption can be made already now.

What should be addressed first is the rigidity of services and the problems this can cause. Services should not be oriented towards specific environments, but should be flexible so that they could be consumed by different users, through diverse communication systems, and with various devices. Pervasive (ubiquitous) computing [98, 9, 113] paradigm addresses this issue by decoupling services, applications, devices and users from each other and viewing them as completely independent entities. They are no longer firmly tied together, but have their own functions and objectives and interact with one another when needed. In particular, applications are seen as special entities that perform specialized tasks on users' behalf. Appropriate infrastructure support allows them to be highly customizable and personalized according to users' needs, roam freely between various devices, adapt to changing environmental conditions and be independent of the underlying communication technology. Similar infrastructure support would be a desirable amendment to modern computing systems as well. Not only would it increase service reusability and improve users' perception of it, but also it would bring current communication standards closer to each other and alleviate the escalating problem of network interoperability.

1.3.1 Abstract Services

In order to make services portable, they should be presented in a rather abstract way, e.g., only general functionality of the service is specified. All of the network, terminal and user-specific aspects of service presentation would not be determined within the service itself, but within the application. The application presents a certain functionality for the user to manipulate the service in the way the user

prefers and in the form his/her terminal allows. Thus, the application concretizes the generic service to a specific functionality and presentation.

1.3.2 Applications' Adaptability

Since applications represent services customized to a form suitable for a concrete network, terminal and user, and since those factors tend to dynamically change, these applications need to be adaptable to possible changes in the environment. Application's appropriate reaction to any occurred changes should preserve quality and usability of the provided service. Adaptability itself means only that the application can change its behavior and knows how to perform, e.g., it may apply a different strategy for achieving its goals, or can reconfigure its own structure appropriately.

Here, an important issue is *proactivity* [98]. It signifies the capability of an application to anticipate future circumstances and adjust its own behavior appropriately and in advance. Proactivity is the way to intelligent self-tuning [41] that further decreases the level of user distraction and promotes invisibility. Capturing user intent is a critical factor in achieving these outcomes since applications seek to foresee user actions as well as environmental conditions.

1.3.3 Context Awareness

Adaptive applications capable of reacting to environmental changes necessitate the use of mechanisms for tracking and interpreting those changes. Such information is referred to as *context* and requires processing to be present in the form appropriate for those applications. Environment monitoring, context acquisition, and context interpretation are generic mechanisms of context-aware systems.

Conventionally three categories of context [33] are distinguished: physical (location, time, etc.), computing (terminal form-factor, battery life, available bandwidth, accessible computing facilities, etc.) and user context (current activity, schedule, intent, etc.) Effective context categorization is a nontrivial problem that is targeted by numerous research efforts [23, 32]. Mature context-aware systems have to carefully collect, interpret, and gracefully integrate various contexts for client applications in order to facilitate their adaptability and smooth operation. A wide spectrum of various approaches [34, 50, 110, 116] that aim at development of context-aware pervasive systems and frameworks has been observed during last decade.

In obtaining context, to achieve good accuracy and simplicity, significant attention should be paid to developing smart spaces. The vision is that smart spaces will soon embrace our daily lives, bringing closer together the computing and physical worlds.

Capturing user intent is a crucial task for today's context-aware systems, even though this is complex to formalize. However, when this is successfully achieved, an unprecedented level of intelligence and proactivity can be achieved. Right now, management of user information profiles is a common way to anticipate user actions. One field that is capable of providing a novel and interesting solution to the problem still remains challenging: *wearable computing*.

1.3.4 Interoperability and Integration

The services populating the computing environment could be combined on demand to create some form of integrated functionality for accomplishing sophisticated tasks. If such functionality can be discovered within the services available throughout the environment, the application can call on the appropriate service and compile the functionality of multiple services.

This is seen as an ultimate outcome of building interoperability on a global scale. Current vision of interoperability is has many facets that reflect different interoperation levels from low-level device and network interoperability to sophicticated semantic interoperability. Some exapmples of low-level interoperability are cyber foraging [8] and Mobile IP [87, 88]. Semantic interoperability [61, 62] is currently one of the most vibrant research fields in distributed computing and especially on the Web. We surveyed some of the approaches to interoperability in our previous work [118]. An outstanding effort has been done in [109] to present a comprehensive conceptual view on existing interoperability issues.

1.4 Intermediate Perspective: Service Portability Framework

In this section we present our original view of a possible intermediate perspective of modern computing evolution. Service Portability Framework (SPF) should be seen as a part of our research work, which contributes to this thesis as an additional motivating factor that guided us in our work and finally led to the "Context-aware Web Service Composition" problem setting.

Service portability is the ability of a service to be reused, in the sense that the service's use is extended to multiple environments. Current understanding of service in communications is rather static and flat. Once created, a service can no longer be altered during its lifecycle. For the development of portable services, two main perspectives must exist. The first is to build portability directly into service. In this case, the service is portable, but is limited to a set of environments and cannot be altered to map other emerging possibilities. In such case the transparency of service provisioning is questionable. Furthermore, service

definition becomes huge with the increase of the number of supported environments.

The other approach is to free the service from any details concerning the environment of its use. In this case, the portability is built into a particular service application instead. Whenever the need for porting the service arises, the appropriate application is constructed or the existing one is modified accordingly. Thus, any particular service application can be created for either a portable or a local use. For each supported environment, the application has a set of operational strategies for appropriate behavior adaptation. Such a service portability framework is a core aspect of the service-centric integration approach.

The main design goal of the service portability framework is to set up a base for seamless provision of any service through any type of environment. By environment it is meant here a particular combination of physical surroundings and computing environment. The latter is the more important of these two aspects because operational characteristics of each environment mainly depend on the available computing facilities and deployed communication standards. By seamless service provision we mean that services are delivered to users in a continuous and distraction-free manner regardless of any changes that may occur in an environment during an active service session. Though such service provisioning paradigm is quite a challenge, and may be unattainable in practice, the objective is to make services as independent of environments as possible. The main idea is not new: to distinguish between two service instances, one of which is unique and as generic as possible, and the other one a highly specific implementation of the service. No matter how the potential environments differ, it should be ensured that the service is always presented by the unique global instance. This instance should always stay unchanged to ensure that the service remains the same while its specific implementations are customized with respect to the requirements of concrete environments.

In order to achieve this goal, it is necessary to introduce a special service provisioning architecture that would separate the global unique service instance from its actual implementation for a specific environment. (Similar ideas were initially proposed by Banavar and colleagues [9]). Such architecture should manifestly adhere to the two-phase principle of service provisioning, where the individual phases are virtually independent and concerned with generic and specific service instances respectively.

It is plain from the definition that the two-phase service provisioning comprises two distinct phases. The first phase is *service creation*. It consists of design and deployment of the global instance of the service. Let us call this unique instance a *generic service*. "Generic" here means that on this stage the service definition is devoid of any specific properties related to environments where the service is intended to be used. The closest practical analogue of a generic service is a Web service, which is relieved from low-level details and is advertised by a

semantic service description that is practically a collection of metadata describing the service. Despite the similarity we intentionally distinguish a Generic service from a Web service to eliminate any association of our conceptual view of a service with its concrete technological realizations.

The other phase is *service delivery*. It embodies a construction of a service application from the generic service created during the service creation phase. The service application is a specific implementation of the service. Since it is aimed for use within a particular environment, all the necessary properties and functionalities are incorporated into it. In other words, every single application of the same service specifically accommodates to the requirements, properties and restrictions of the concrete environment. It must be noted that at this stage, while the application might be altered, the service still remains unaltered. Having been constructed and launched, the application should continuously stay tuned to the requirements until its termination. However, there is a tendency of environmental conditions to progressively change, which makes the process of delivering service applications more complicated.

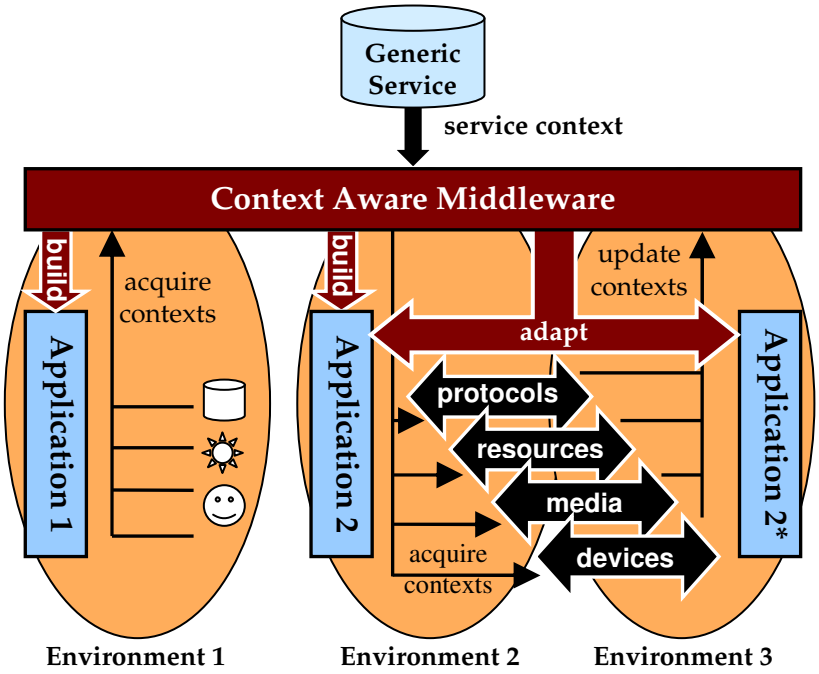

FIGURE 1 General view of the service portability framework

The concept of service portability implies not only that every generic service may obtain a multitude of differing applications in various environments, but also that environments and their properties may dynamically change during service delivery. As long as such changes render service applications inadequate or even inoperative, the applications should be adapted at run-time not to disrupt pending service sessions.

The function of applications' adaptation is performed by a specific middleware for service portability (as well as by the applications themselves or in collaboration between the applications and the middleware). We propose context-aware reflective middleware architecture for service portability. Context-awareness plays an important role here as it is a good foundation for application adaptation frameworks and as it allows an even treatment of environmental conditions along with other contexts. Reflectivity [55] is a vitally important property for such middleware, since it allows capturing dynamics, making the architecture viable in a highly dynamic mobile environment.

Figure 1 shows the service portability framework in a general setting. When a user or some application invokes a service from the service provider, the context-aware middleware receives a description (which we call "service context" to emphasize that such description is treated similarly to other contexts) of the corresponding generic service located in the provider's network. Service context can be categorized as a subtype of computing context and specifically describes requirements and restrictions of the corresponding generic service to be applied when building applications based on it. The middleware also analyzes various context information acquired from the target environment, and builds a new service application based on the results of the analysis made. As we have attempted to show in Figure 1, in this way the middleware can build different service applications (Application 1 and Application 2 in Figure 1), which are capable to operate on top of specific platforms, for different environments from the same generic service and configure them appropriately. All the necessary information influencing the application being built is obtained from within the context acquired from the target environment. However, the environment where the application is running can suddenly change. For example, the user may move to another device, or hand over to another environment with different transmission medium and protocol stack; some resources in the environment may vanish or their levels may change. Though this list is far from complete, all these changes (shown in Figure 1 with thick black arrows) essentially mean that the environment has changed. These environmental characteristics are retrieved by the middleware in the form of context at run-time. At this point the application may turn inoperative, since the environment has changed. To prevent that, the middleware has to perform a run-time adaptation of the application (or the application can adapt itself if it is able to do that). The middleware detects context changes, analyzes them and performs an adaptation procedure on the application. In the simplest case the middleware changes the set of application's interfaces to the computing environment to comply with the new environment, to which the application has moved. If the application has been initially provided with all the necessary interfaces, it adjusts them on its own.

1.4.1 Service Creation

As it was already noted above, a generic service is designed at the service creation stage of the service provisioning procedure. The obvious objective of this design effort is to supply the service definition with all the necessary data to make the service instance unique, unambiguous and easily interpretable. The generic service instance should be unique in the sense that it should be distinguishable from any other service both in terms of identification and in terms of usage. At the same time the service definition should not be overloaded with any redundant details, which might, in the worst case, turn the service unsuitable for certain environments. For example, if a generic service specifies to which network protocols an application should bind, such application cannot be produced for an environment where a network technology implemented does not support the specified protocols. A generic service basically contains only metadata about remotely exposed functionality of the service and about its operational requirements. Properties related to network standards, protocol stacks, data formats, transmission characteristics, and device capabilities are application-specific and should be implemented within an actual service application. In Web Service Architecture such metadata is called service description [15]. For example, it may provide a link to the service's location, list restrictions on the capabilities of end terminal equipment, store some authentication information, etc. Metadata contained in a generic service is to be used during the service delivery stage to construct an appropriate service application which will deliver the service to an end user in the most consistent and reliable way.

Such a generic service design is beneficial from several points of view. First of all, it allows maintaining a unique globally available service instance, which is easily recognizable and is not likely to be confused with any other service entities. Secondly, this instance is really generic, which means that it does not possess any implementation-specific properties. Being generic, it does not shrink the range of possible specific implementations of the same service and increases potential reusability of the service. Finally, due to separation of the service and its application, the generic service design allows service porting between diverse environments without any alterations to the service itself.

One more attractive opportunity offered by the generic service design is service composition. In such service provisioning framework two or more services can be jointly delivered by a single application, which is rather simple to build. Instead of integrating two specialized applications, which in many cases are not at all composable, we can compose two corresponding generic services in a rather simple but formal way and then build an application on top of the composed generic service. However, complex interaction of two services at run-time would still present quite a complicated problem.

1.4.2 Service Delivery

Service delivery phase begins with the construction of the service application and ends with its termination. This phase can be logically split into two sub-phases: load-time and run-time. At load time, i.e. build time, the application is constructed from the generic service and is brought into conformity with initial conditions (context) ascertained from the environment. This way, the application is particularly adjusted to operate on top of certain terminal capabilities, protocol stack, resource level, etc. Thus, it is guaranteed that the newly launched application does not fail to perform correctly at startup. At run-time the application runs as normal, but it adapts or gets adapted whenever essential environmental changes occur. Essential environmental changes can be understood as changes in the environment that may influence or even adversely affect the application's performance and operability. Run-time adaptation of applications leads to preservation of service session continuity, which is indispensable for service quality and robustness.

In order to create applications in a rather automatic manner, i.e. without real human designer's intervention, a special framework for application design is needed. An attempt to create a theoretical foundation for such automatic service application design was made in our previous work [119]. In essence, this research work proposes a service reference model for characterization of services with respect to different functional layers, which exist in the system and implement certain functionalities related to services. The main concern of the elaborated model is to create an accurate layering of service-related functionalities so that they could be developed relatively independently.

1.4.2.1 Service Reference Model

The proposed service reference model [119] is rather generic and follows the Open Systems Interconnections (OSI) network reference model [121] design, and therefore represents layered framework for service classification. Vertical service distribution is the fundamental base for the model. However, the optimal choice of the actual scale layers is not easy. They should be selected to present a functional division that is not too detailed or too loose, but universal. Here *universal* means that the model could describe any existing or applicable service. The utility of such universal representation should be seen in a possibility of unified modeling of all services, which would specifically assist us in porting services between heterogeneous environments. Such service reference model would allow layer-by-layer testing of a service with respect to requirements of an environment to which it is to be ported, and consequent adjustment of the service structure on a particular set of layers.

Figure 2 [119] presents a view of the entire service reference model, comprising the network, transport, middleware, application, content and metacontent basic service scale layers. The lower layers of the model act as a base for higher ones. The service-related functionality varies from the most specific on the bottom layers to the most generic on the top ones.
Service layers contain basic service building blocks. They should be non-overlapping to carry exclusive service-related functionalities. Each of the layers focuses on different service consumers. The service consumers for each service scale layer are indicated in Figure 2.

Metacontent is the content "above" content. It provides metadata about the service and its content. Metacontent strives for increasing service *availability* in communication networks and facilitates intelligent and automated service discovery and interpretation. Public search engines and intelligent agents are the intended consumers of the metacontent service layer.

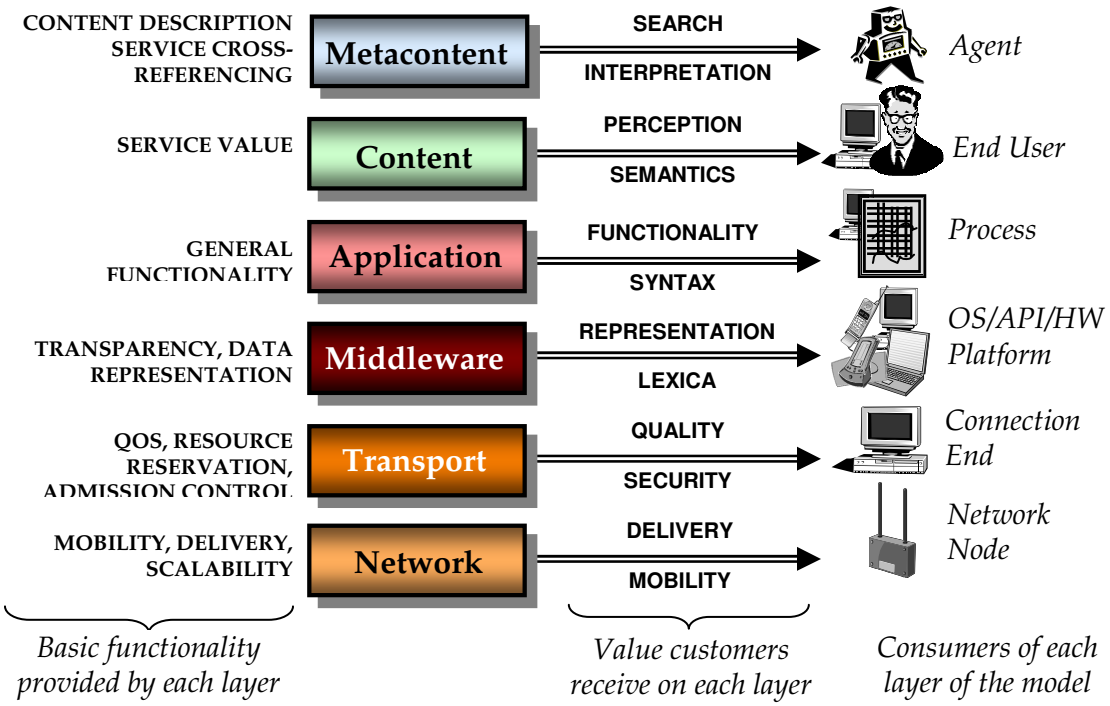

FIGURE 2 Service reference model

Content represents the specific service value to be provided. Be it information, rights, or any other kind of resource, the content comprises a set of content elements designed to create clear user perception of the value being provided. Based on such perception the end user, who is the service consumer of the content layer, restores the semantics of the information value. Content may be of any type, such as video, voice, data, etc. The main requirement is content *consistency*, i.e. no meaningless, discrepant or obsolete content should be provided.

Application presents the core service facility, which is implemented in a software format, is typically distributed because it is intended for networked environments, and is capable of complete rendering of the service content. It acts as a full-fledged service instance customized for the environment. The application is based on the set of API functions provided within the OS platform and renders the content syntax on top of available lexical primitives.

Middleware is used to denote a set of generic services above the operating system. Typical middleware services include directory, trading and brokerage services for discovery, transactions, persistent repositories, and different transparencies, such as location transparency and failure transparency [91]. Extending the definition, middleware represents an intelligent mediator between the hardware/network platform and the human-oriented services. The primary functions of middleware are transparency, context-awareness, and application adaptation. According to [91], context-awareness will be a fundamental property of future mobile applications. As stated in [42], context information can be used to facilitate communication in human-to-computer interaction.

Transport is an end-to-end vehicle for the service delivery over a communication network. The transport layer primarily takes care of traffic categorization, QoS provision and end-to-end transmission security. The service must be categorized into appropriate traffic types in order to receive the required quality. Once assigned, the values of the bit rate, delay, packet loss and other QoS parameters must persist along the entire transmission path on the network. Traffic characterization must account for the specifics of every single service in order to provide reliable transport and to utilize efficiently the resource and admission control.

Network layer mechanisms work in a point-to-point fashion and play mainly an auxiliary role supporting the transport layer functionalities. Load balancing, routing, scheduling and flow control allow for increased service transport robustness. The primary interest here is mobility tracking and location management. These techniques provide the base for the creation and deployment of mobility-based and mobility-support services within communication systems.

The designed service reference model offers a fundamental base for clear service characterization. *Service scale* and *service platform* [119] aspects are taken into account for a realistic service description, and result in a differentiation between vertical and horizontal services. The concepts of vertical and horizontal services were previously described in our work [120] and in the earlier work by Lehtonen and Harju [64]. Vertical service encodes a service instance that has a rigid hierarchical vertically integrated structure. According to our specific viewpoint, horizontal service stands for a functionality provided on a single layer of network system architecture. The proposed model has a layered architecture corresponding to the vertical distribution of horizontal scale layers, which reflects the logical separation of service-related functionalities. Vertical services are composed of

service scale primitives provided by horizontal services. The primitives are layer-specific and determine the functionalities utilized by vertical services on each layer.

The model allows the service to be platform independent. This property makes the model eligible to meet the network interoperability challenge. Middleware hides the specific network details from services and their creators. To make services compliant with concrete network environments, middleware adapts service applications at the run-time. We call this process as *service portability*.

Separating creation of services from their delivery is of critical importance for ubiquity of service provisioning. Along with horizontal division of vertical service structures this aspect facilitates loose coupling of business description of services (in the form of generic services) and their technical specification and creates a foundation for building interoperability among contemporary distributed computing environments.

1.4.2.2 Transition from Vertical to Horizontal Services

Current services are vertical services. This means that they are tightly built into applications that deliver them. They are developed with "all-in-one" principle in mind. All the service-related functionalities from user interface to content rendering are rigidly built into the service instance. The service is seen as a monolith having an indivisible structure, which is rather implicit than explicit. Such an approach is inflexible in the sense that the service is adaptable only to the extent that is stipulated already at the stage of service design. To illustrate this comment, let us take an example of mobile telephony. A user cannot normally cross the border between two different communication networks (by different we mean different network standards) without interruption to the active call, because the application, which delivers the calling service to the user, is not designed to be operated on top of a different protocol stack.

An alternative view on the service design focuses on transition from rigid vertical services to horizontal ones. Horizontal services are those provided by the environment. They bring certain system functionality to applications. For example, transport service provides functionalities such as packet transmissions, end-to-end security, traffic management, etc. Applications utilize these functionalities to achieve their specific goals. However, whenever some of the available horizontal services change (like in the example with GSM telephony), applications are no longer capable of pursuing their own goals, having been intentionally designed to operate on top of a narrow set of horizontal services. In the modern communications world with its strong integration trends, such a state of affairs appears increasingly unacceptable. Instead, horizontal services should be effectively used to alleviate the efforts on application design and to make service applications more flexible.

According to our idea, the structure of an application should be modular. It should comprise several functional layers similar to those described in [119]. Each layer may contain multiple modules that perform specific functions. However, these modules should not necessarily be predetermined at the stage of application construction. Instead, the structure has to be flexible so that modules can be added, removed or substituted at any time. They are to be tied together by some sort of unified interface, which would allow for certain variety of structural alterations. These modules are called *service primitives* and should be mostly provided by the environment where the service application happens to be operating. The role of the modules is not to perform a certain functionality, but to utilize a corresponding functionality of horizontal services available in the environment, and to transform this functionality into the form in which the application needs it.

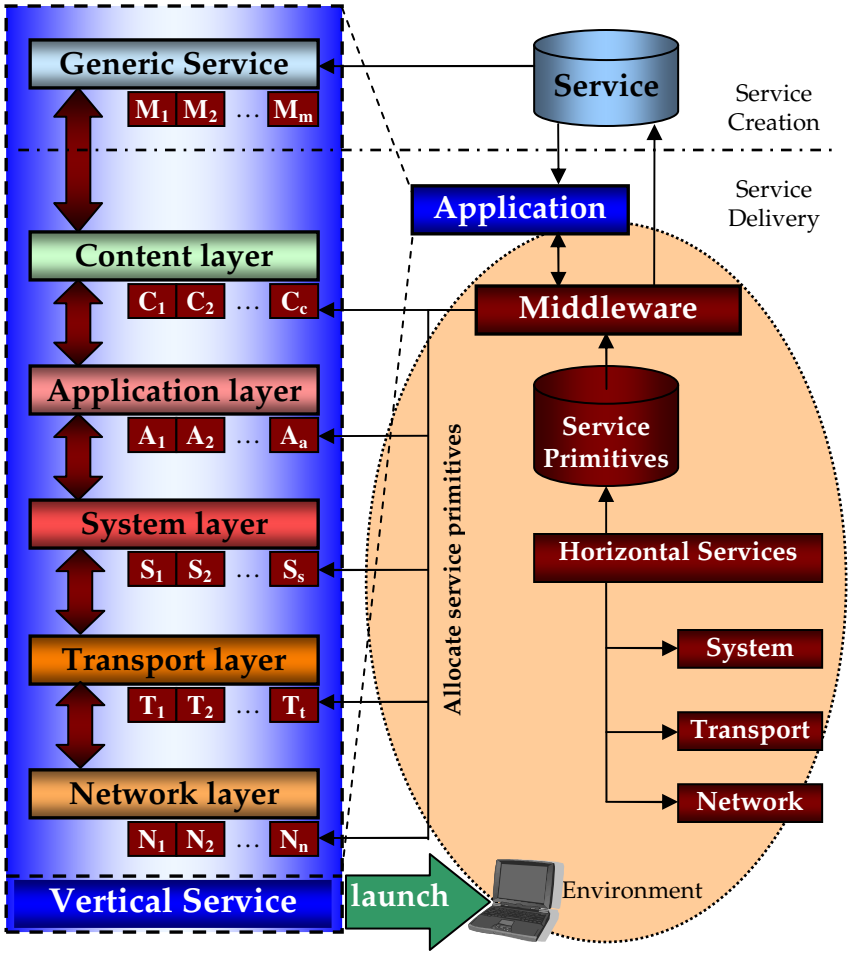

FIGURE 3 Two-phase service provisioning architecture

Programmatically, service primitives can be seen as *adapter design patterns* [44] or *wrappers* [101], reusable object-oriented software components. These patterns can be combined to collaborate with one another and implemented as software applications on top of the actual communication platform using special

frameworks (which are provided by middleware in our case). Pattern-oriented software design is a mature programming approach and has been successfully implemented in various communications software solutions. However, here we abstract from the details of this approach application. Detailed discussion of pattern-oriented software design and architectures can be found in [44, 102, and 21].

Let us explain in a greater detail how service portability framework works. Figure 3 depicts the scheme of dynamic assembling of a vertical service application from service primitives available in an environment. To illustrate this scheme in a rather practical way let us use the following scenario.

Steve is at home and is going to leave soon for a lecture which takes place at the university campus. But he knows that a televised football match he has been waiting for the whole day is to start in 5 minutes. He takes his handheld device, which has a wireless local area (WLAN) connection to the Internet, and through the web interface of the corresponding TV channel's site invokes a video streaming session to watch the broadcast of the match on the way to the university. In the provider's network there is a generic service that provides a connection to the TV broadcast server. The middleware sends a query to this generic service and gets the service context of the video streaming service. This context data contains the address of the TV broadcast server, authentication information for establishing a connection to it and some technical requirements that should be met in order to receive the video streaming service. After that the middleware acquires the context from the environment, namely the device capabilities from the profile of Steve's handheld, characteristics of the active wireless link, traffic management schemes supported by the underlying system architecture, protocol stack used in the WLAN network, etc. Having analyzed this context information, the middleware starts the process of application construction. It locates necessary service primitives with respect to the results of context analysis made and assembles them to get an application. For example, the middleware may obtain video and audio content primitives from the broadcast server to make the application capable of processing and outputting streaming video and audio. It also obtains, from the local environmental repository, the primitive which allows the application to receive packets of streaming traffic type. It is important that the local WLAN network enables horizontal transport service that supports the streaming type of traffic with appropriate Quality-of-Service (QoS) policies. Similarly, the necessary primitives for all the layers shown in Figure 3 are found. Once all the primitives are allocated the vertical service application is finalized and launched on Steve's handheld device. Steve takes his device with him and watches the broadcast on the way to the university.

Thus, a vertical service no longer has a fixed structure. Instead, it is adjusted to the concrete environmental requirements by adding appropriate service

primitives. This way a multitude of specific implementations of the same generic service can be created.

Portability of a service in an active state, i.e. a service being delivered to a user at that moment, is achieved in a very similar way. At the application's run-time the environment may suddenly change due to the user's switchover to another environment. Having found itself in another type of environment, the application may turn inoperative, which means that some of the currently equipped service primitives are, perhaps, no longer suitable to deliver the service through the new type of environment. Therefore, the application must, accordingly, be adapted to correctly operate in the new environment. The adaptation procedure, which is to be applied in this case, is essentially a substitution of invalid service primitives with valid ones. New primitives are located within the new environment and forwarded to the application. The application reconfigures its structure with the new service primitives and starts operating as normal. The adaptation procedure is primarily controlled by the middleware.

Continuing the previous example, soon after leaving home Steve also leaves the range of his home WLAN connection. His handheld device automatically switches connection to a wide area GPRS network available in the new outdoor environment. The middleware immediately recognizes that changes have taken place in the context and determines that the video streaming application is no longer capable of operating in the new environment. It collects contexts from the new environment, analyzes them and determines which service primitives in the application's structure clash with the new requirements. For instance, the middleware may discover that a different protocol stack is used in the GPRS network. Hence, packets of different format should be received by the client. The middleware contacts a local repository containing service primitives and substitutes all the obsolete primitives within the application with the operational ones. This way the application is being ported to another environment without being terminated and re-launched. In the best of the cases Steve would not even notice the switch between environments. But, generally, some slight distraction period may be evident if the volume of context data is quite significant for fast processing. In the case of non-real-time or asynchronous services strict continuity of a service session is not important. In these cases packets can be buffered by the middleware somewhere within the system for a later retrieval by the application. The end result of all this is that, without taking any additional actions with his handheld, Steve can watch the TV broadcast while sitting on the bus on his way to the university.

It should be noted that a change of an environment does not necessarily imply that a user has handed over to a different network system. It may also indicate that he has switched over to a different terminal, and an application should immediately "teleport" to the user's new device to preserve the service session from being prematurely disrupted. Some significant changes in the

environment, such as, for example, base station crash or resource saturation, will also lead to re-selection of service primitives in order to adapt the application's performance to these new operational conditions.

1.4.3 Service Portability Scenarios

In order to illustrate how the service portability framework operates, let us consider a few of the most general operation scenarios. Home network is understood as a network where the service originally resides. Foreign network is a network to which the service is supposed to be ported.

The first scenario corresponds to the case in which a user at a foreign network requests a service (see Figure 4 (a)). This case may be named *static*, since no efforts for run-time adjustments of the service application are needed. In the most general case, applications are constructed and launched on demand. So far, the user's request for a service must result in an appropriate service application launch in the foreign network. Assuming a generic service design, the application can be issued in a foreign environment in the same manner as it was intended for a local use. Thus, a static service portability scenario illustrates the importance of a generic service design.

Figure 4 (b) illustrates a *dynamic* service portability scenario. A user in the home network requests the service and then moves to another network. The application is constructed and launched while the user is connected to the home network. But when the user's terminal makes a handoff to another network, the application is perhaps no longer suitable for service delivery because of probable incompatibilities between the platforms. One alternative way to resolve the situation is to terminate the current application and to issue a new one in the foreign environment. The main flaw of this option is discontinuity in the service session. This may be unacceptable to the user, who expects seamless service delivery.

To preserve access transparency and service quality, the application should be adapted to a new environment instead of being re-launched. In this case, a service session persists during the handoff and the user is relieved of possible service terminations. Being handed over to another environment, the portable application senses the environment and selects an appropriate strategy or reconfigures itself.

The dynamic service portability scenario addresses the situation of Single Terminal and Multiple Environments (STME) with the terminal mobility involved, whereas the static scenario corresponds to Single Terminal and Single Environment (STSE) without mobility. However, there is a case of multiple terminals (MTSE and MTME) with the user mobility involved.

The MTSE case is basically irrelevant to the integration problem. Therefore, only the MTME case is considered (see Figure 4 (c)). The scenario is quite distinct

from the STME from the adaptation point of view. The application must not only adapt to another environment, but also find a terminal to which the user switched and adapt to that terminal. So, the application gets two execution contexts: network and terminal.

The "follow-me" principle for tracking user mobility in pervasive systems has been previously used for developing Teleporting applications at Olivetti Research Lab [13]. This experience can be used as a foundation for portable "follow-me" applications.

Portability of service applications across diverse networks becomes possible because of applications' adaptability. The outcome of generic service design is the existence of a unique service instance that needs no alteration to be reused among heterogeneous environments. The opportunity for service adjustment is provided under the control of the applications. This can also facilitate service integration inside an application. The last but not least issue to be addressed is adaptation frameworks for portable applications.

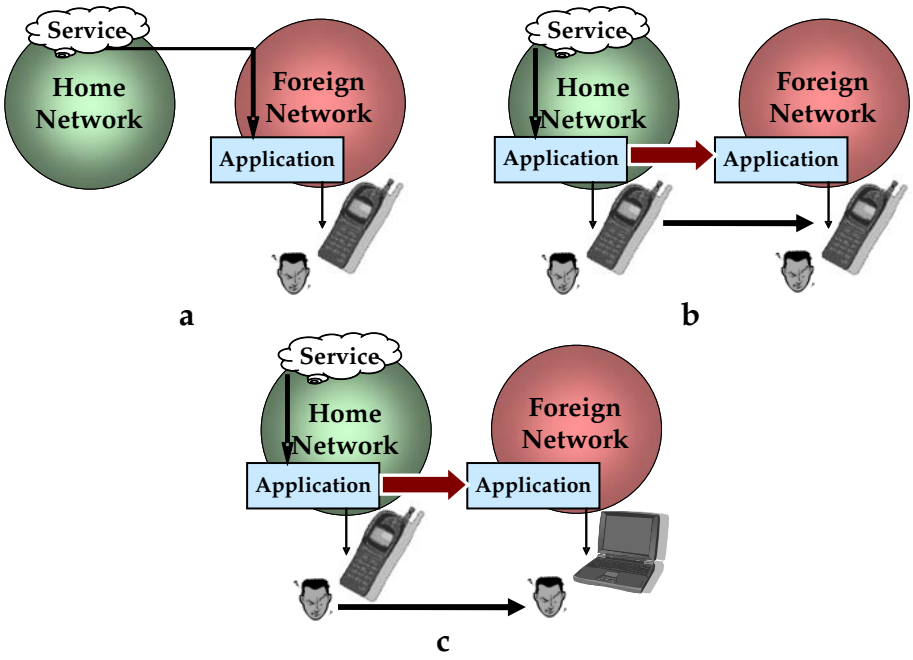

FIGURE 4 Service portability scenarios

1.4.4 Adaptation of Service Applications

The separation between service creation and service delivery necessitates considering a service not as a "black box" but as an object that has an internal structure and properties. Although the service obtains only an abstract description at the stage of creation, it is concretized to the form of application on the delivery

stage. The service is processed several times to adsorb the specific properties that correspond to the context. The initial abstract form of the service is seen as a rough core that needs appropriate shaping. In coming across with different contexts, such as terminal form factor, network bandwidth, etc., the same abstract service may result in a variety of distinct service applications. Such a vision is more complex than the current service provisioning frameworks, but more flexible. Once created, the service can be modified dynamically with respect to the momentary changes in conditions found in the current environment.

The core element of the service portability framework is the context-aware middleware. It is designed to perform actual porting of service applications. The middleware is in charge of service adjustment at application's load-time and of application adaptation at application's run-time. However, application-specific issues cannot be sensibly managed by the middleware. They require application-aware adaptation, in which the middleware assists the application. The types of supported adaptation procedures are described in the subsections below.

In the proposed service adaptation framework all the data which are used as a basis for adaptation are treated as *context* for the purposes of uniformity and simplicity. This would allow collecting and storing all the necessary information in a common repository and inferring adaptation decisions in an easier way.

Various contexts are collected by numerous context providers, such as sensors, monitors, and such special sources as user and equipment profiles. Physical contexts, such as location, signal strength, levels of interference, etc., are measured by sensors. Diverse network contexts, e.g. resource levels, are collected by monitors. Service context is retrieved from metadata provided within the generic service. User context can be obtained from user profiles. All the collected contexts are examined by the middleware and stored in the database. The way of modeling context for this infrastructure is a complex issue, especially taking into account such a broad definition of the utilized contextual information. According to the survey of context modeling techniques made in [106], ontological representation of context is superior in many aspects to other formal modeling methods. This approach is especially attractive when applying it on the Web for contextualization of Web Services. In the Web domain, there are some interesting and powerful approaches to ontological context modeling, among them C-OWL [16]. As it will be seen below, our ultimate vision of service portability is particularly focused on Web Services and Semantic Web Services, the underlying technology and capabilities of which provide extra motivation to apply ontologies for context modeling.

We only consider functional organization of the middleware infrastructure. Architectural layout of its distributed composition is not yet described in detail. However, we will briefly describe the principle of our distributed composition. The proposed middleware architecture consists of three main functional blocks:

- Context Acquisition Proxy

- Context Manager
- Context Reasoning Engine

The middleware manages three data repositories:

- System Contexts
- Context Model
- Service Primitives

Context Acquisition Proxy controls the population of context providers. It collects readings from sensors/monitors at their report rate and sends updates to the context manager. Acting as a proxy, this facility filters received readings. It discards all inessential changes in monitored contexts using certain implicitly given criteria for their estimation.

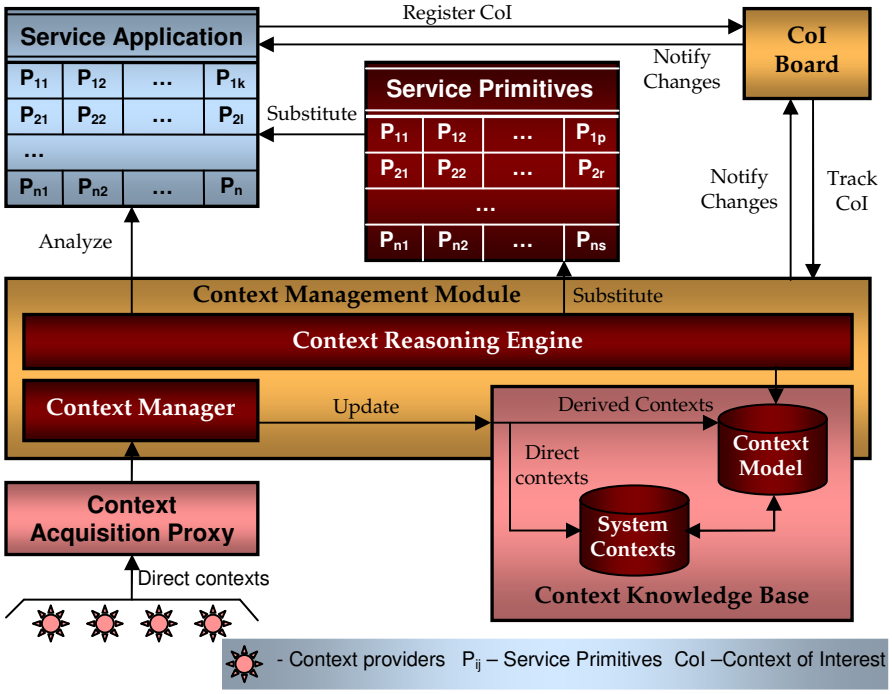

FIGURE 5 Service adaptation framework

Context Manager is responsible for maintaining the context model in a consistent state by updating it promptly with essential context changes. It also controls context information exchange between different architectural entities and manages context dissemination procedures.

Context Reasoning Engine is the core part of the middleware infrastructure that infers adaptation decisions. Its apparent role is to perform reasoning on the context. The engine checks for changes in the context knowledge base looking for possible conflicts between the application and system contexts. A conflict represents a certain discrepancy between the application's operational properties and the

current environment parameters. This discrepancy would form a hindrance for the application to operate properly in the observed conditions and must, therefore, be resolved during the adaptation process. Whenever the context reasoning engine determines such a conflict, it starts reasoning on the base of the context model trying to deduce a satisfactory solution for the detected conflict. The reasoning procedure may or may not cause a launch of an adaptation procedure. Adaptation, as an ultimate way of conflict resolution, usually results in a substitution of application's service primitives that clash with occurring contextual conditions. On the basis of the obtained reasoning decision the engine selects new primitives from the corresponding repository and instructs the application to substitute them (see Figure 5).

System Contexts Repository is a relational database that stores the whole variety of context variables monitored in the system and their current values.

Context Model Repository contains an ontological model of contexts present in the environment. It may express both physical attributes and logical properties of those contexts. The model is capable of formalizing complex relationships between different contexts. It is maintained in the actual state and gets updated as necessary by the context management facility.

Service Primitives' Repository stores an array of service primitives compatible with a current environment. The primitives are built in correspondence with the horizontal services that the environment provides. These primitives are basically pieces of executable program code. They are shaped as programmatic patterns and can be linked to each other through a set of unified interfaces (frameworks). The primitives are constructed by special middleware services, which remain out of the scope of this thesis. If some substantial changes occur to the environment and influence its horizontal services, the corresponding primitives need to be removed from the repository and new primitives added. The service primitives' repository is used in the adaptation procedure, whenever the middleware decides that some primitives of a certain application cannot operate in the current environment. In this case, the middleware makes a decision about which primitives should be selected from the repository to substitute the application's obsolete ones.

From the architectural viewpoint, the middleware is composed in a distributed fashion. The main architectural entities are *Context Management Module* that comprise context manager and context reasoning engine facilities, and *Context Knowledge Base* that consists of context model and system contexts repositories. These two modules are centralized. Context acquisition proxies are allocated locally for groups of context providers present in certain parts of an environment. Data exchange between local proxies, central management modules and context-aware applications is based on a certain protocol. A survey or any justification of such protocols is beyond the scope of this thesis. The operation of the middleware architecture is synchronized by means of event triggering system, which allows capturing unexpected situations in the environment.

The operation of the described middleware architecture depends on what type of adaptation procedure is being currently accomplished. This adaptation framework is sufficiently flexible to allow three main types of adaptation strategies to coexist within it. The first alternative is the adaptation of the service quality with respect to the available resources. Such an adaptation strategy is handled by the application that treats current resource levels as contexts of interest and adapts the quality of its output whenever corresponding contextual changes occur. Another adaptation strategy can be applied in the environments supporting the resource reservations. This alternative is effectively handled by the middleware to relieve the applications from Quality-of-Service (QoS) related technical details. Finally, the most sophisticated alternative strategy is *user-aware* adaptation. It features a corrective action for the user to perform [98]. This option appears useful when quality is critical, and the system cannot currently accommodate it, but a solution that requires user intervention can be proposed.

1.4.4.1 Background Adaptation

Background adaptation is an adaptation procedure that is entirely handled by the middleware without the application's assistance. The application under adaptation may not even be aware of the adaptation performed.

This is a default type of adaptation. The middleware always tries to perform it first, and only if the detected problem cannot be resolved by the middleware, it would proceed to other adaptation procedures.

Background adaptation is generally utilized when inappropriate service primitives of the application belong to the layers that correspond to the horizontal services provided by a local environment. For example, if a user has made a handover to another network system which uses a different protocol stack for transmitting packets, or different QoS framework are utilized, then certain service primitives on lower layers of the service reference model may appear inappropriate for a new environment and consequently turn the application inoperative. The middleware is capable of detecting and settling such a problem by its own strength, since transport and lower system layers are transparent to the application. So, the middleware reassigns the primitives of transport and network layers imperceptibly to the application.

The role of the context reasoning engine is to analyze the configuration of the application, and the service context, and to detect which service primitives of the application are obsolete. After that the engine assigns, to the application, new service primitives from the local service primitives' repository.

1.4.4.2 Application-Aware Adaptation

Application-aware adaptation is an adaptation procedure in which the application is aware of the adaptation being made and makes the final decision on how it is to be adapted. This type of adaptation is carried out by the application with a possible assistance of the middleware.

Application-aware adaptation usually takes place when inappropriate service primitives of the application reside on the layers that are not related to a local environment but to the service itself. Metacontent, content and application layers [25] obviously belong to this category. For example, after Steve's handover to a GPRS environment on his way to the university campus it may appear that the system cannot accommodate the quality of video streaming service due to overall scarce bandwidth. The middleware has already failed to satisfy the requirements of the service by performing background adaptation. Therefore, the adaptation has to be made by the application to reduce the quality of the delivered service. In the above example, the application should follow the recommendation of the middleware and stop using the video content primitive which is too resource-demanding, and keep the audio content primitive only in order to deliver the TV broadcast in an accurate and error-free manner.

This type of adaptation cannot be handled by the middleware on its own because the service primitives, which need to be substituted, are not transparent to the application and comprise its core functionality. Furthermore, new primitives for substitution cannot be retrieved from a local environment in this case because of their absence in the local service primitives' repository. They can only be provided to the application in advance at load-time (or retrieved from the service's origin), so that the application could adapt itself in critical situations. The role of the context reasoning engine here is to detect the conflict, and to attempt to find a solution on its own - after the failure of the background adaptation to analyze what kind of application adjustment is required for a successful solution - and finally to propose a deduced solution to the application for a subsequent accomplishment.

Besides that, the application may be given some adaptation strategies at the construction stage. It may be required to adapt its behavior during its operation with respect to certain changing factors. If the application can perform self-adaptation, it is usually called *adaptive*. The proposed middleware architecture facilitates the support for adaptive applications by providing them with a possibility to register their contexts of interest (CoI) within a special tracking facility called CoI board. When an application registers its CoI within the CoI board, necessary contexts start being tracked by the middleware (if it is technically possible) and the changes are reported directly to the CoI board. The board then notifies the application about the changes in its CoI, and the application can adapt its behavior as necessary.

The benefit of such a framework is in that the application may register quite complex contexts, which cannot be measured directly, but only inferred on the base of the context model. Such complex contexts are deduced by the context reasoning engine, thus relieving the application from sophisticated computations and giving it the possibility to adapt itself almost effortlessly.

1.4.4.3 User-Aware Adaptation

User-aware adaptation is the most complicated type of adaptation. It implies that the final adaptation decision is made consciously by the user because neither the middleware nor the application succeeded in providing a satisfactory service quality level on their own.

Nevertheless, all the work to find an appropriate adaptation decision is done by the middleware and the application. In case they succeed to find any reasonable solution, they propose it to the user, who finally decides what to do. Such a solution, if it can be found, is called *corrective action* [98]. By performing this corrective action the user ensures the required level of service quality.

Let us use the previous example to illustrate this idea. After Steve's handover to a GPRS network system it appears that the available bandwidth is too scarce for a satisfactory quality of video streaming service. The middleware fails to solve the problem due to a low physical capacity of wireless links. The application fails to reduce the quality of the delivered service, because the video component of the broadcast appears critical at the moment (Steve manually set in advance the application's options to indicate that video would be critical to display). However, having analyzed the system and user contexts, the middleware and the application find a corrective action soon after Steve gets out of the bus: if he moves to the nearest lobby, which is situated 30 meters away from his current location, Steve will get an acceptable quality level of video transmission, because the lobby is equipped with a WLAN hot-spot.

In principle, the middleware is capable of deriving corrective actions without any assistance of the application. However, only the application is able to communicate the corrective action that is found to the user.

1.4.5 Service Portability for Web Services

The service portability framework is described in the previous sections in a rather broad way lacking any details about possible applications of the presented vision. The main intent for such broad description is to show that the framework does not in principle depend on specific environments and/or services.

Web Services is one of the most elaborated implementations of service-oriented architecture [84], which promotes interoperability at the highest level. The Web Service concept closely resembles our vision of generic services. Therefore,

the application of the described approach to industrial Semantic Web environments is currently the most suitable and justified way to go about. First of all, the Web Service Architecture [15] does itself provide flexible support for two-phase service provisioning. It can be easily seen that our view on service provisioning resembles Web Service provisioning framework with minor amendments. These differences should not be applied to web services themselves, but can be realized within the middleware infrastructure.

Another technical motivation for this approach is the use of Semantic Web ontologies for modeling context. There are several major approaches to context modeling [106], and they have their own strengths and weaknesses. Among these approaches ontological context modeling is currently the most reasonable method due to the following properties of ontologies:

- high flexibility and manageability;
- possibility for distributed composition;
- capturing incomplete and ambiguous information;
- high level of formality;
- applicability to existing environments.

Furthermore, some of the context sources in a real enterprise environment may already provide contextual information in an ontological form (e.g. user and equipment profiles, service descriptions), which makes it even more logical to use ontological approach for modeling the rest of the context.

Finally, Semantic Web ontologies (in particular, those in the OWL format) provide means not only for context modeling but also for contextual reasoning, which significantly alleviates the implementation efforts for building context reasoning engines. Ontological approach to context modeling has already justified itself in similar research [16, 24, 25, 47, 63].

From a practical viewpoint, the described infrastructure for service portability is easier and more reasonable to implement and deploy within a single enterprise network rather than on a wide scale for public communication systems.

A typical service provision scenario for an industrial Semantic Web environment is presented in Figure 6 [120]. Service Provider and Service Requestor are generic entities that represent the owner of a Web Service and its consumer respectively. These entities can express relationships of business-to-business type as well as business-to-customer type. Here we do not need to concern ourselves with the process of service discovery.

This scenario differs from the generic case illustrated earlier in this thesis in that high-level contexts are already represented in the ontological form within the enterprise system. The Web Service corresponds to the notion of Generic Service, and the service description, advertised by it, is a concrete instance of service context or meta-context. Low-level contexts, such as time, spatial coordinates, connection characteristics, etc. are received as usual from low-level context

providers and transcribed to the ontological view by Context Management Module with respect to the available ontology structures in the system Enterprise Ontology. Enterprise Ontology describes the entire system in a comprehensive manner, annotating properties of its elements and relationships between different entities within the environment.

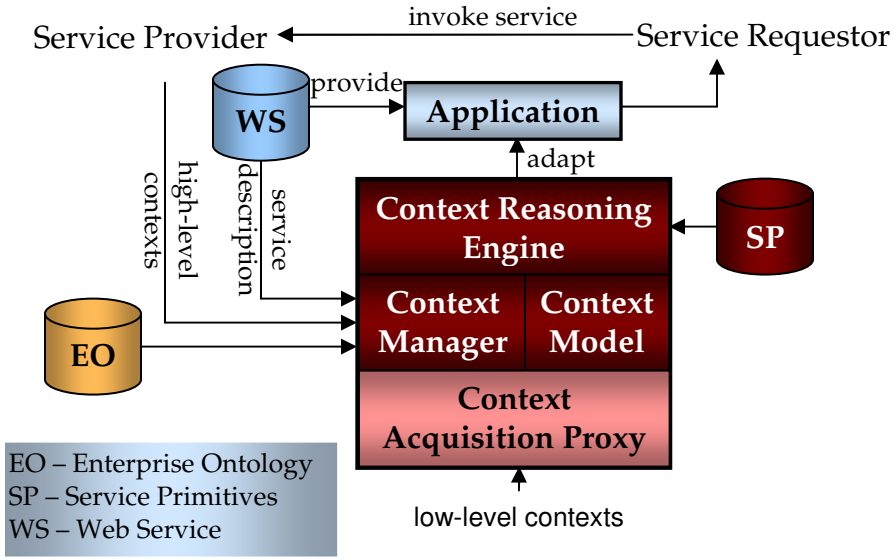

FIGURE 6 Service provisioning scenario in industrial Semantic Web environment

Let us consider the following example. As soon as Steve arrives in the university campus he realizes that he does not know the place where the lecture is being held. He browses to the university's website and invokes the location-based campus guidance service. He submits a query for a lecture location to the service and gets an answer which indicates that the lecture will be held in the campus building owned by a special unit of an industrial company collaborating with the university. This unit has its own enterprise environment covering the building. As soon as Steve enters the building he is classified by the internal security system as a participant of the visiting lecture and authorized to use certain enterprise services available within the system. At this point the guidance application is adapted to operate on a possibly new technological base present in the enterprise network (new wireless access standards such as WLAN and Bluetooth, new protocol stack, QoS frameworks, etc.) and to use internal location-based Web service providing visitor guidance inside this particular building. To do this the middleware acquires the service description of the corresponding Web Service, analyzes the context model of the environment, and the local user profile, performs reasoning and finally extracts necessary service primitives from the environment to properly re-configure the application. Following the instructions on the screen of his handheld Steve takes an elevator to the third floor. The guidelines provided to Steve are the result of a context reasoning procedure performed by the new context-aware

application and based on Steve's location inside the building. Steve's location is its specific "context of interest", which is tracked by the middleware on the application's demand. The path is constructed and locations are identified with respect to the enterprise ontology, which exists in the system and defines such relationships as, for example, "containment" to enable reasoning about location.

Bringing context on top of semantics in Web Services is an attractive feature of the framework that would allow Web Services to be more flexible and proactive. The major benefit that Web Services can obtain from utilization of Service Portability Framework is in mobility. Applications based on Web Services would be able to move freely between communication environments and user terminals without being interrupted and re-launched. It should be also noticed that the framework has potential to deal with the problem of Web Service composition by using context awareness for customization of composite services.

1.4.6 Concluding Remarks

The rapid proliferation of telecommunications reveals a solid trend towards technological divergence, which can be detrimental to the interests of the parties involved. Convergence is highly desirable in order to introduce a global communication environment. The pervasive computing paradigm is an attractive viewpoint on computing and communications, it helps identifying a long-term goal for communication network integration and establishes a number of crucial challenges to be solved in the immediate future. As a compromise, a short-term approach to deal with these challenges outlines complex integration issues on multiple system layers. The extensive integration approach is oriented toward the interoperability of currently existing communication environments; additionally, there are many useful ideas that already can be borrowed from the fertile vision of pervasive computing. Implementation of those ideas will definitely enhance the efficiency and robustness of the current integration approaches and improve the usability and transparency of today's mobile computing and communication systems, thereby pushing them to a new qualitative level.

Service portability is the core concept of this service-based integration [118, 120]. Portable services can be reused across various communication environments. To develop portable services, specific service design within environments must be avoided. The separation of service creation and service delivery allows for a single generic service instance in the interconnected environment. Adaptable applications play an essential role within the service portability framework, since they can perform services' porting across environments. Assigning this function to the application complements the generic service design. Applications customize the services to the various environmental contexts. The services then are launched on demand and can adapt at run-time, without service interruption.

The most important thing about the framework presented is that it not only makes context-aware services available to current mobile users, but also proposes a context-aware approach to build up interoperability between today's communication systems.

The Service Portability framework is based on the idea of decoupling service applications from actual services. Such approach simplifies service design, since services no longer require to be endowed with application-specific details. This, in turn, increases service reusability, allowing any service to be consumed through any type of environment without being modified or reissued. The portability of services is achieved by introducing adaptable service applications that can be reconfigured. These applications, instead of services, are adapted whenever environment undergoes any significant change. Such principle allows hiding any unnecessary details from service creators and managing run-time adaptation locally with the particular service application. Sophisticated efforts on remote service adaptation are not needed and service scalability is preserved. To manage the framework we specify a context-aware infrastructure, which captures dynamic environmental changes in an efficient manner and provides reconfiguration and adaptation mechanisms for service applications. It utilizes horizontal services provided by environments to build service applications in a more pertinent way and as a result makes service concept more flexible and open.

1.5 Problem Statement

The main objective of the thesis research is to design a formal framework for utilizing contextual information in a Web Service composition process. This would allow Web Services to be composed in a more consistent, dynamic and efficient way. Here the important goals are:

1. Formal modeling of Web Services and context-aware Web Services;
2. Web Service composition planning in a context-aware fashion;
3. Context-aware dynamic adaptation of composition structures during execution.

Planning procedure is of paramount importance for Web service composition process. Creating accurate and deadlock-free composition plans helps seamlessly integrate business processes and create robust value-added electronic services for a consumer. Utilization of contextual information for building composition plans can enhance the service composition process with respect to its efficiency, robustness and flexibility and further alleviate the efforts for composition planning and reconfiguring on the fly. Context-aware composite services can become superior to ordinary Web services in terms of quality, relevancy, usability and scalability.

Bringing context to Web services in general and to Web service composition process in particular is a fresh and fertile idea, which is rather weakly addressed within present-day research. With the proper formalization approach context will be a new dimension along which the service composition process can be adjusted and compositions can be controlled. Should context be incorporated into composition planning frameworks, these would gain in correctness, adaptivity and verifiability. Incorporating pieces of specialized context-aware functionality directly into composite services will allow building more dynamic and responsive services, which adapt themselves with respect to changing context via reconfiguring their structure on the fly.

1.6 Solution Methodology

The core formalism of the Web service composition planning methodology to be designed in the thesis is the apparatus of Petri nets. Petri nets are basically a process modeling technique featuring formal semantics. It is a suitable graph-based tool for pictorial and abstract conceptual modeling of complex Web services representing partially ordered set of business processes, sub-processes and atomic operations. The idea is to modify the Petri nets technique and extend it with additional context elements for more elaborate process modeling and more flexible control of composition patterns.

The Petri nets formalism possesses some outstanding qualities to motivate such particular selection of service composition modeling tool. More specifically, its applicability to service composition is proven efficient, it offers powerful capabilities for modeling state-transition systems (which Web Services basically are), it establishes a high level of formality and a rich but concise notation, finally it creates models capable of self-analysis and verification. The presented methodology starts with simple conventional Petri net patterns representing workflows at their basic level and ends up with a sophisticated custom type of Petri net models, called Meta Petri nets, for representation of entire context-aware composite Web Services at the highest modeling level.

1.7 Contributions

The essential contributions we make in this thesis represent our original research ideas and results. They are intended to serve as specific solutions to achieve the goals established in Section 1.5. Here we briefly describe our contributions.

The main contribution of the thesis is Context-aware Web Service Composition framework, an integral formal solution to our major challenge of context utilization for Web Service composition.

The particular contributions solving the major design goals 1-3 (see Section 1.5) are respectively the following:

1. Declarative service algebra for composition of Petri net based Web Services' process models;
2. Specification of goal-driven context-aware planning procedure for Web Service composition;
3. Formal Petri net-based approach for modeling and reconfiguration of Context-aware Composite Web Services.

For the most part this thesis describes and discusses these particular contributions in a very detailed fashion. Nevertheless, we will present a compressed survey of the contributions made and explicitly describe how they solve the established goals.

1.8 Expected Results

The major expected research result of this thesis is the detailed and comprehensive design of the Context-aware Web Service Composition framework supplemented with a formal description of methodological approach to Web services representation and operational semantics.

More specifically, the expected results include but are not limited to:

- Selection of workflow composition patterns for representing composite Web services (as part of the contribution 1 (see Section 1.7));
- Creation of a set of algebraic operators for synthesis of Web Services (as part of the contribution 1 (see Section 1.7));
- Determining tools for structural analysis and verification of Web service composition (as part of the contribution 1 (see Section 1.7));
- Definition of Context-aware Composite Web Service (as part of the contribution 1 and 3 (see Section 1.7));
- Detailed description of context-aware service composition planning stage (as part of the contribution 2 (see Section 1.7));
- Detailed description of context-aware service composition reconfiguration stage (as part of the contribution 3 (see Section 1.7)).

The main research question to be answered in the thesis is the following:

How can context be utilized in the process of Web Service Composition to improve relevance, robustness and quality of composite Web Services?

In order to answer this overall research question, we set up the following detailed research questions, which are to be answered based on the achieved results:

RQ 1: What are Web Services? How are they formally modeled and composed?

- How are Web Services formally defined? How Petri net formalism can be used to model Web Services?
- What kinds of Petri net structures are needed to model Web Services in a process-oriented, precise and unambiguous manner?
- How can Petri net models of single Web Services be formally combined to model a composite Web Service? What kinds of additional Petri net structures and elements are needed to facilitate such combination?
- How can Web Services be analyzed and verified using Petri net models? How can Web Service composition process benefit from utilization of Petri net formalism?

RQ 2: What are Context-aware Web Services? How are they formally defined?

- How can Context-aware Web Services be formally defined using Petri nets?
- How does modeling of Context-aware Web Services differ from that of ordinary Web Services?
- What are composite Context-aware Web Services?

RQ 3: How can context utilization enhance Web Service composition process?

- Can context enhance functioning of goal-driven composition planning procedures?
- Can context improve quality and accuracy of component service matchmaking?
- Can scalability and reliability of Web Service compositions be increased via context utilization?

RQ 4: How are Context-aware Web Services composed?

- How can context assist in building dynamic, reconfigurable Web Service compositions?
- What is the difference between Context-aware Composite Web Service and context-aware composition of a Web Service?
- What kind of Petri net formalism and structures are needed for robust modeling of context-aware reconfigurable Web Service compositions?

- How can context reasoning be performed in real-time to facilitate reconfiguration of composite Web Service models? How are reasoning procedures modeled using Petri nets?

1.9 Thesis Structure

The dissertation is structured to provide a logical, well-ordered, detailed and coherent presentation of the Context-aware Web Service Composition framework and complementary work stages.

The thesis consists of 5 logically and functionally separated Chapters:

Chapter 1 "Introduction and Motivation": In this Chapter we presented our vision about the modern stage of distributed computing and made some insights regarding its further evolution. We described our previous research work and ideas as motivation for the chosen research problem of Context-aware Web Service composition. Finally, we formulated our main research problem by establishing specific research goals to be achieved, determining solution methodology, identifying main contributions to be made, previewing research results to be expected, and setting up the particular research questions to be answered.

Chapter 2 "Background and Related Work": this Chapter describes background knowledge necessary for understanding of the subsequent Chapters. Here we provide definitions and descriptions of core and most influential concepts and ideas in the areas of Service-Oriented Computing, Web Services, Semantic Web, Web Service Composition, Context-aware Computing and Petri Nets Theory. We also survey the state of the art in the Web Service Composition research, to obtain a foundation for clear understanding of the essence and current trends in Web Service Composition and to reuse existing achievements in our work. Finally we briefly describe how our versatile approach combines the described concepts and ideas to offer an innovative and comprehensive formal solution to the problem of Web Service composition.

Chapter 3 "Petri Nets for Web Service Composition": this Chapter shows how Petri nets formalism is utilized for modeling of Web Services. More specifically, it gives a process-oriented definition of a Web Service, specifies Petri net instrumentation for construction of Web Service models, and investigates, in depth, possible

workflow patterns for orchestration of composite Web Services. As the main outcome, this Chapter provides a specification of declarative Web Service algebra that describes composite Web Services' Petri net models as algebraic expressions and provides algebraic means for their analysis and optimization. Thus, Chapter 3 presents the contribution 1 (see Section 1.7) of this thesis.

Chapter 4 "Context Utilization for Web Service Composition": this Chapter provides several important results. It specifies the entire automated goal-driven Web Service composition process. It discusses context-aware refinement of this process and formally describes contextualization of its component procedures. This Chapter presents a formal definition of context-aware Web Service and principles of its modeling. It further specifies the run-time context-aware Web Service reconfiguration procedure, which is the most innovative idea of the entire research work. Chapter 4 provides detailed view of contributions 2 and 3 (see Section 1.7) of this thesis.

Chapter 5 "Conclusions": this Chapter concludes the thesis with a summary of the achieved results and made contributions. It provides detailed but compressed answers to the research questions in Chapter 1. It emphasizes main benefits and advantages of the proposed solution. Finally it identifies major directions of research and development work continuation.

2 BACKGROUND AND RELATED WORK

This Chapter describes background knowledge, concepts and ideas, and existing research achievements and results relevant to the problem of Context-aware Web Service Composition. The knowledge of the concepts presented in this chapter will produce a clearer understanding of the ideas expressed in the contribution Chapters 3 and 4.

Chapter 2 is organized as follows. We start in Section 2.1 with a discussion of the Service-Oriented Computing paradigm identifying its main principles and impacts to modern views on distributed computing. In Section 2.2 we survey Web Services technology, the most elaborate implementation of service-oriented computing paradigm. Section 2.3 is devoted to the discussion of an influential vision of a semantically-enriched version of the Web referred to as the Semantic Web. Section 2.4 offers an extensive survey of the state of the art in the research field of Web Service Composition and attempts to devise a structured view of the Web Service composition process, which we will consequently use in Chapter 4 to define our Context-aware Web Service Composition framework. In Section 2.5 we describe the basics of the Petri nets modeling formalism and its specific modifications, which we utilize for Web Services' modeling in Chapters 3 and 4. Section 2.6 describes the main concepts in Context-Aware Computing, which is fundamental to the understanding of the ideas we express in Chapter 4. Finally, in Section 2.7 we conclude the Chapter with an explanation of how the described technologies and techniques can be brought together to design an innovative solution to the Web Service Composition problem.

2.1 Service Oriented Computing

Service Oriented Computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications [84].

To build the service model, SOC relies on the Service Oriented Architecture (SOA), which is a way of reorganizing software applications and infrastructure into a set of interacting services [85].

SOA has its roots in Component Based Architecture and enhances it to accommodate contemporary electronic business needs related to openness, integration and management.

SOA is, in particular, seen as an evolution of several system, software and networking paradigms, namely, the Component Based Architecture, Object Oriented Design and Distributed Systems.

A Component Based Architecture utilizes the principle of functional division. It splits the entire complex functionality into simpler functions and encapsulates these functions into separate architectural components. Distributed Systems paradigm enhances this principle by further distributing the components over some physical location scale.

Component Based Distributed systems are widely implemented at the present time as various enterprise intranet solutions as well as over the Internet. The major drawbacks of such systems on the current stage of distributed computing evolution are their rigid and proprietary technology support and poor interoperability. However, the good features of the Component Based Architecture, which SOA inherits from it, are reusability and repurposing of system components.

The reference definition of SOA is given in the OASIS technical specification, Reference Model for Service Oriented Architecture [65], and is the following:

Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.

Although the above definition is taken from the standard, it is still a rather loose way of stating what exactly constitutes SOA.

A more general definition of SOA is given by Nickull [81]. According to it SOA is an *architectural paradigm for components of a system and interactions or patterns between them such that a component offers a service that waits in a state of readiness and other components may invoke the service in compliance with a service contract*.

Despite the fact that specific implementations of SOA may significantly vary they are always build on top of the following major concepts:

- Services (also referred as service implementations);
- Service descriptions;
- Service operations (also referred as service interactions).

In terms of SOA the concept of services has a variety of definitions. In the most general of these Nickull [81] defines a service as a *contractually defined behavior that can be implemented and provided by a component for use by another component*.

Perhaps, the most famous and more technical definition is given by Papazoglou and Georgakopoulos [84]: "services are self-describing, open components that support rapid, low-cost composition of distributed applications." From our viewpoint the most interesting aspect of this definition is direct assignment of services to perform composition.

In his other work [85] Papazoglou also defines a service as a business function, which is implemented in a software format and supplied with a widely intelligible formal documented interface.

Generally speaking, services always act to accomplish certain functions. They are owned by organizations, which are collectively called *service providers* and expose their functionalities over the Internet on the basis of a standard set of protocols. Services feature a service interface which is based on open standards and languages. What is more, services establish a distributed computing infrastructure that facilitates both intra- and cross-enterprise application integration because they communicate over the Internet and are provided by different enterprises.

Service consumers (or clients) can be other enterprise applications, external applications or users.

In order to perform everything claimed above, services must be subjected to the following major requirements:

- *Technology neutrality*. Services must not rely on or be bounded to concrete implementation technologies and standards utilized at client and service sides. Instead services have to be operated by the mechanisms which comply with widely recognized standards.
- *Loose coupling*. No knowledge of client and service sides' internal structures should be a prerequisite for seamless services' operation.
- *Location transparency*. Services must be exposed to the Internet in a way that allows ubiquitous invocation of them irrespective of their physical location and the client's location.

One more interesting requirement, which is not however a major one, is *flexible configurability* of services which manifests in that services should be configured late in the process of service provisioning and allow for changing their configuration dynamically.

Some of the well-known implementations of SOA-compliant services are Web Services [15], J2EE [107] and .NET [75].

Service descriptions are standardized, separately provided definitions of services. They are used for advertising services to potential clients allowing them to understand such issues as what the particular service does or how it can be

invoked. More specifically, a service description basically comprises the following specific descriptions of a service:

- *Service capability description* identifies the purpose of the service usage, its objectives and expected outcomes it produces as a result. This is a semantic type of description, which is only interpretable using some specific conceptual taxonomy (e.g. a semantic ontology) of the terms used in the description.
- *Service interface description* represents the service invocation details, which are syntactic parameters such as service inputs, outputs and associated messages' formats.
- *Service behavior description* describes the intended behavior of the service during its execution, for example, in the form of workflow process.
- *Service quality description* lists various Quality of Service (QoS) characteristics of the service such as cost, performance parameters, security attributes, integrity, reliability, scalability and availability.

Common standards for service description are vitally important to ensure the openness and robustness of SOA. Currently available mature standards for service description are Web Service Description Language (WSDL) [26] and ebXML Collaboration Protocol Profile [82]. Recently, with the growing demand for and popularity of Semantic Web services several semantic service description standards have appeared. We discuss them later in this Chapter.

One more concept vitally important for a proper understanding of SOA is *service advertising and discovery*. Service advertising refers to the process of communicating service descriptions to prospective consumers of corresponding services, while service discovery is the counterpart operation denoting the procedure of obtaining information about a service, its properties and terms of invocation by a potential consumer.

Service advertising and discovery can be organized in different ways, which are not stipulated within the most general definition of SOA. However, there are well known and widely used methodologies such as *Pull*, implying that the consumer requests service descriptions directly from service providers, and *Push*, which encodes the procedure of sending service descriptions to prospective consumers by the service provider. The combination of these two methodologies known as the *publish-subscribe* pattern is apparently the most popular advertising/discovery solution due to its increased scalability and reliance on neutral third-party agents for service advertising and discovery.

Having described the major SOA concepts and entities, we can now present the basic Service Oriented Architecture.

As described in [85], the basic SOA defines an interaction between software agents as an exchange of messages between service requesters (clients) and service

providers. Clients are software agents that request the execution of a service. Providers are software agents that provide the service.

The responsibility of providers is publishing descriptions of the services they provide, while clients have to discover the descriptions of the necessary services and consequently bind to them on the basis of stipulated invocation terms.

Thus, the basic SOA is an architecture which is based on the interrelationship of three distinct entities: the service provider, the service consumer (requestor/client) and the service discovery agency (registry/repository). The allowed interactions between these entities are *publish*, *find* and *bind*. The corresponding graphical representation of the basic SOA is shown in Figure 7.

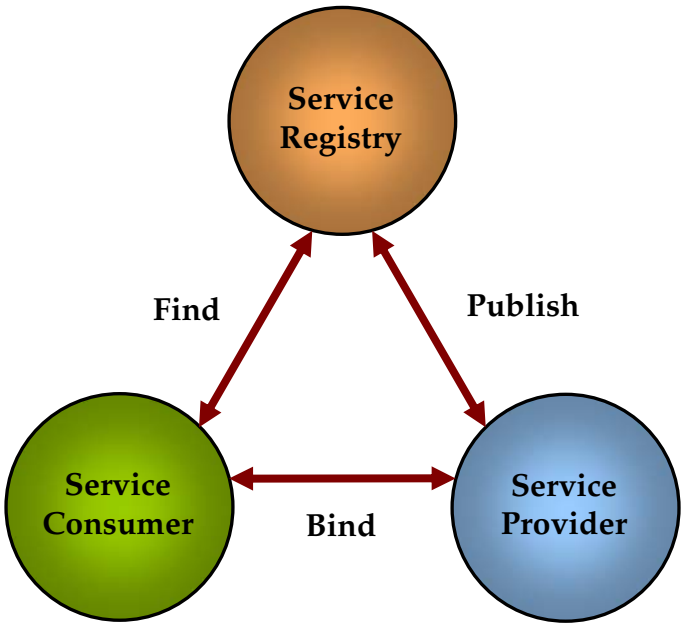

FIGURE 7 The basic Service Oriented Architecture

The principle of such SOA operation is the following. Whenever a service provider has a software module accessible through the network which it wants to expose for ubiquitous usage by external consumers, it creates a service description for this service implementation and publishes it to an available service registry. As soon as the service description is accepted to the registry the service becomes discoverable by external consumers. A service consumer can discover this service description using the "find" operation and retrieve it from the service registry. After that the consumer uses the obtained service description to bind directly to the corresponding service provider and invokes the service implementation offered by it.

Note, that service provider and service consumers are logical constructs, which means that a business entity can simultaneously play the roles of both the service provider and the service consumer.

The two most prevalent standards of service registries for discovery and advertising of services are currently the OASIS Universal Description and Discovery Interface (UDDI) [27] and the OASIS ebXML Registry-Repository [83].

Although basic services, being implementations of complete business functions and technically decoupled from their providers and consumers, provide outstanding means for efficient reuse of applications, they still support composition of integration-ready applications only passively. In order to provide active support for service composition the basic SOA has to be extended with dedicated composition mechanisms.

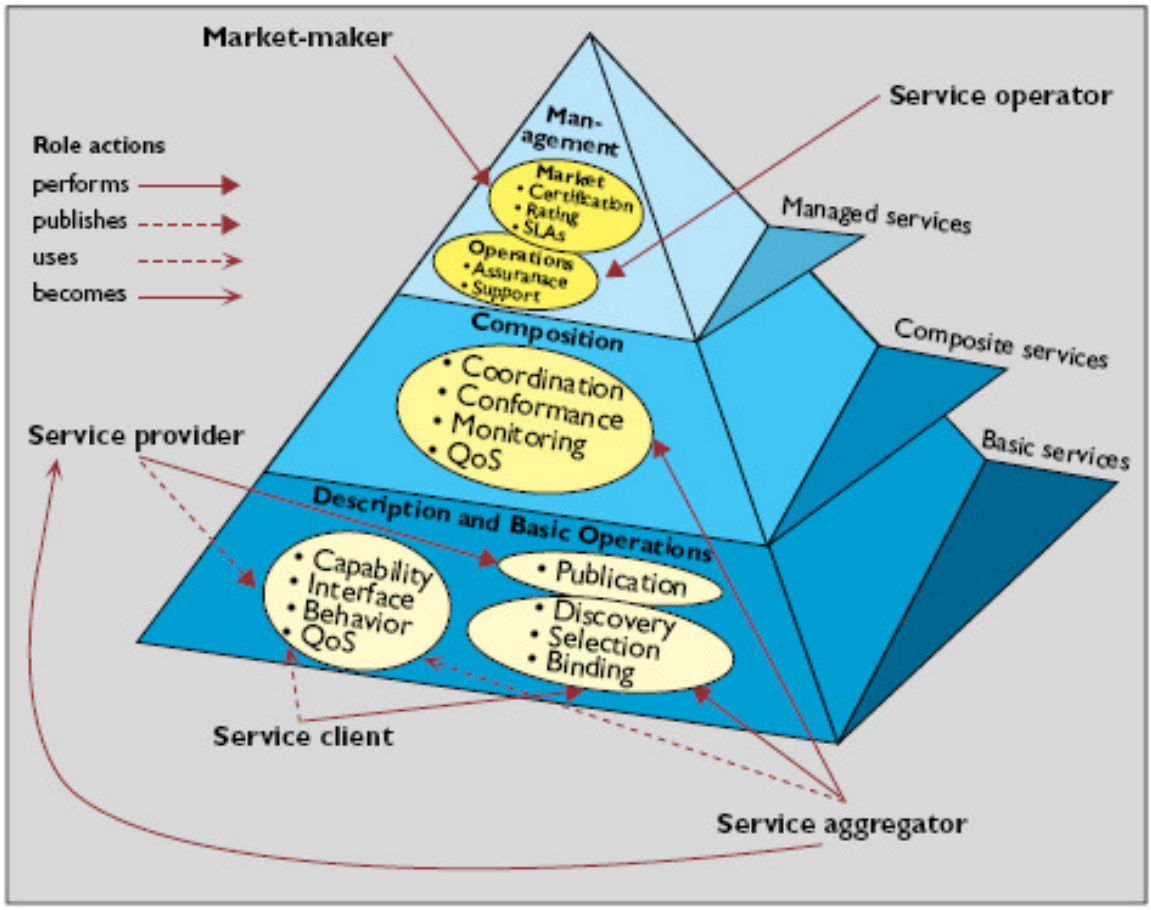

FIGURE 8 The Extended Service Oriented Architecture

The extended SOA (ESOA) is an architecture that addresses such crucial issues as service composition and service management. This is a layered architecture, which adopts the basic SOA and all its constructs as its foundation on the bottom layer. The graphical representation of ESOA taken from [84] is shown in Figure 8.

It can be seen from Figure 8 that The extended SOA (ESOA) is the architecture that addresses such crucial issues as service composition and service management. This is a layered architecture, which adopts the basic SOA and all its

constructs as its foundation on the bottom layer. The graphical representation of ESOA taken from [84] is shown in Figure 8.

It can be easily seen from Figure 8 that the ESOA adopts two additional functionality layers on top of the basic SOA layer. These layers are composition and management.

We are not interested in the service management layer functionality because it is basically out of the scope of this thesis. We just note that the service management layer is supposed to provide a variety of high-level business functions from enterprise platform management and transaction management to performance assessment and support for open service marketplaces.

The service composition layer is what is precisely in the scope of our interests. The service composition layer encompasses the necessary roles and functionality for the consolidation of multiple services into a single composite service [84]. The service composition layer introduces a new role of *service aggregator*. Service aggregator is an entity which performs the actual composition of services into a composite service. Service aggregator is similar to service consumer in the sense that it discovers service description of several services, selects the appropriate ones, and binds to them. But instead of simply using their functionality service aggregator composes these pieces of functionality in a new service. This new composite service should also exposed to the potential consumers by publishing its corresponding service description for further discovery. So, service aggregator must first create such service description for the composite service and then publish it, thus, accepting the role of service provider.

What is important, service aggregator should be able to compose both basic and composite services into the composite service being constructed. In this way recursiveness of the composite service definition and a higher degree of service reusability are achieved.

In addition to the described functions service aggregators develop certain specifications and/or code for accomplishment of additional functions such as the following as described in [84]:

- *Coordination*: controls the execution of the component services, and manages dataflow among them and to the output of the composite service (by specifying workflow processes and using a workflow engine for run-time control of service execution).
- *Monitoring*: subscribes to events or information produced by the component services and publishes higher-level composite events (by filtering, summarizing, and correlating component events).
- *Conformance*: ensures the integrity of the composite service by matching its parameter types with those of its components, imposes constraints on the component services (to ensure enforcement of business rules), and performs data fusion activities.

- *Quality of Service (QoS) composition*: leverages, aggregates, and bundles the component's QoS to derive the composite QoS, including the composite service's overall cost, performance, security, authentication, privacy, (transactional) integrity, reliability, scalability, and availability.

So, it can be easily seen that ESOA already stipulates an impressive bundle of functionalities for the support of rapid and efficient creation of composite services, which conform to the main SOC principles. ESOA thus presents a new form of services' reusability not in the role of complete business functions/solutions, but as proprietary pieces of functionality to be included into the larger functionality as primitive components.

2.2 Web Services

As we mentioned in the previous section, Web Services are one of the mature and, perhaps, the most elaborate implementations of the SOC paradigm. So, the technical specification of Web Services thoroughly follows the principles of SOC presented above with a distinction that it binds the application of Web Services to a bundle of well-recognized protocols, languages and other standards. Therefore, we will not repeat what we have already said about Web Services in the context of SOC. Instead, here we mainly focus on the technological aspects of Web Services, including a brief description of the underlying standards.

Since a Web Service is seen as a specific class of SOC-compliant services, it is obvious that all the definitions and descriptions presented in the SOC section and related to SOC-services stand for Web Services as well.

Web Services are a particular solution that allows efficient expansion of distributed computing principles over the Internet. They actually can figure as the main building blocks of the Internet's distributed computing environment. The central point in the Web Services paradigm (just as in the case of generic SOC-services), which can be considered the main driving force of Web Services development, is interoperability. Web services help developers expose their applications running on a variety of platforms and frameworks to the Internet and ensure their transparent consumption and interoperability with other applications and services. This is achieved by using a standardized XML-based interface for description and communication.

Apart from interoperability being the corner-stone of the paradigm, the advantages of Web Services are impressive, especially in comparison with more traditional approaches using architectural middleware for building interoperable software. According to [111] these advantages consist of:

- *Easy and fast deployment* of Web Services via combining and reusing of existing services.
- *Just-in-time integration* meaning that a service that is effectively decoupled from other system's components and hardly sensitive to the changes can be configured and integrated quite late in the service provisioning process to reflect up-to-date characteristics of and collaborations among the individual system's components, so presenting fewer possible points of failure.
- *Reduced complexity by encapsulation* meaning that a particular service implementation is not important when compared with the provided service's functionality. This latter behavior is basically taken into account during service consumption and creation of an appropriate functionality while the details of service implementation are paid no attention to.

The W3C's Web Services Architecture specification [15] gives the following technical definition of a Web Service:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Web Services are provided over a session layer on top of a variety of common Internet transport protocols such as HTTP, SMTP, FTP and TCP/IP sockets. As it is stated within the given definition, Web Services additionally use a stack of dedicated standards which operate on top of common transport protocols. These standards are:

- *Simple Object Access Protocol (SOAP)* for message exchange among services, service providers and service clients.
- *Web Service Description Language (WSDL)* for specifying service descriptions.
- *Universal Description, Discovery and Integration (UDDI)* for service advertising and discovery.

Web Services are also often called XML Web Services, since they use eXtensible Markup Language (XML) [19] for serialization of messages and encoding of data, whereas Web Service descriptions represent XML documents.

XML plays a fundamental role in Web Services computing. To comply with the principles of Service Oriented Computing described in Section 2.1 and ultimately to provide means for interoperability of services and applications, data and messages, which are to be exchanged among participants of Web Services provisioning process, have to be presented in a platform-neutral yet interpretable format. By providing an efficient platform-independent data encapsulation

framework, XML becomes an excellent candidate for defining a platform-independent data exchange protocol. It uses the XML Schema Definition (XSD) [38] for validation of XML documents structure and XML namespaces [18] for binding XML documents to specific schemas, thus ensuring interpretability of XML documents at the endpoints of data exchange process.

SOAP [78] is a standard communication protocol for Web Services. SOAP provides means for message exchange between participants of Web Service provisioning process and for performing remote procedure calls over the Internet. SOAP is not a transport protocol. Therefore, it provides no functionality for data exchange between endpoints. Instead, it utilizes a lower transport protocol to carry out such an exchange. This transport protocol is typically HTTP, but can also be SMTP or other Internet transports. However, HTTP is perhaps the best choice, since it is a globally recognized, widely supported and hence the ubiquitous transport protocol on the Web.

On the other hand, SOAP is based on XML. It uses XML serialization for encoding data within SOAP messages. Thus, the data a SOAP message conveys are always interpretable at either endpoint.

HTTP and XML are thus the two corner-stones which make SOAP totally platform-independent. As the result, SOAP is independent of the platform, operating system, programming language and object model.

SOAP operates in a well known "request-response" fashion. It specifies two distinct types of messages, Request and Response, to allow service clients to put request for remote procedures and service providers to respond to such requests respectively.

We do not intend to describe the structure of SOAP messages, since this lies outside the subject of the thesis. Nevertheless, we should mention that one of the main strengths of SOAP, in addition to platform-neutrality, is its remarkable simplicity and ease of implementation, characteristics which make SOAP so popular when implementing interoperable applications on the Web.

Whenever a client wants to use a Web Service, it must exactly know what this particular Web Service does, where it is located and how it can be invoked. In other words, the client has to use certain interface to invoke the service and it needs to understand how to use this interface. To alleviate clients' efforts and to extend the range of potential service clients, the service has to be described in a standard manner and the description made available on a global scale. We consider availability concerns later, and focus now on service descriptions.

Not much can be added to the discussion of service description we made in Section 2.1. Web Service descriptions basically comply with the requirements by SOA on service descriptions. The standard description language for Web Services is W3C's Web Service Description Language (WSDL) [26]. It is used to describe Web services and their corresponding interfaces. Not surprisingly, WSDL is based on a globally interpretable XML-based grammar. A Web Service description

written in WSDL defines the service as a collection of the communication endpoints, which are capable of performing message exchange or remote procedure calls. WSDL descriptions generally consist of two distinct parts: abstract and concrete. The abstract part defines the endpoints and message formats that can be used during service invocation. The endpoints or *ports* are described by defining their *port types*, the abstract collections of allowed *operations*. The concrete part of the service description gives a specific representation of ports by *binding* their abstract operations and messages to specific network protocols and data representation formats. A WSDL description may contain multiple bindings of abstract port descriptions, which promotes its reusability. Thus, a port is defined by associating a network address with a reusable binding, and a collection of ports defines a service [111]. The possibility of importing modular WSDL descriptions to other WSDL descriptions even further promotes their reusability and flexibility. The extensible WSDL framework allows binding of WSDL definitions to specific communication frameworks, e.g. SOAP, HTTP, etc.

Despite the described flexibility of WSDL descriptions, WSDL is still a somewhat limited service description framework from the viewpoint of SOC. The problem is that basic WSDL descriptions provide only service interface description. Service capability, behavior and quality descriptions are not basically presented in WSDL documents. Service capability is described in WSDL using only loose human-interpretable natural language description, which is of little help when a WSDL description has to be automatically processed. To cope with this problem, WSDL descriptions are made extensible, so that they can be easily complemented with additional descriptions such as, for instance, the OWL semantic service capability description and the BPEL4WS workflow service behavior description.

As we have seen so far a standardized approach to service communication and description implemented with the Web Services framework enhances service consumers' experience in terms of service accessibility and usability and also allows electronic business to benefit from rapid integration at reduced costs. However, in order to effectively sell their outstanding capabilities, Web Services have to provide a mechanism for exposing them for consumption on a wide scale over the Internet. Such a mechanism is the final missing corner-stone for forming the complete basic Web Services architecture. A discovery agency primitive, which we described as a part of SOA architecture, can play that exact role. It is usually called *Service Broker* when applied to Web Services. A service broker is essentially a service registry that facilitates the publishing and discovery of Web Services.

A common framework for Web Services advertising and discovery is OASIS Universal Description, Discovery and Integration (UDDI) [27]. UDDI is an XML-based, platform-independent, global and open framework for businesses to discover each other, share their business information and communicate. UDDI is often called the "yellow pages" of Web Services. UDDI is implemented in the form of business registries, which contain structured information about various business

partners and applies "register once – publish everywhere" advertising principle. A UDDI registry entry describes a business and the services it offers. The entry can be logically split into three parts: the "white pages" listing address and contact information of the company, "yellow pages" providing the industrial classification of the business based on standard taxonomies, and "green pages" describing technical information related to the service, e.g. service interface. Technically, the UDDI XML schema defines four main data constructs: *business entities, business services, binding templates* and *tModels*. Business entities describe information about businesses such as their names, contact information, descriptions and services offered. Business services describe specific services and can be associated with multiple binding templates, being entries for service invocation. A tModel is associated with binding templates and represents a technical service fingerprint describing the specific bundle of protocols and frameworks that this particular service utilizes. The tModel may already contain a WSDL description of the corresponding Web service, but it is generally sufficiently flexible to describe any kind of service.

UDDI also provides a programmatic interface for interaction with UDDI registries. This interface contains message specifications for performing allowed interactions with registries and includes two distinct APIs. Inquiry API allows locating businesses, services, bindings or tModels within the registries. Publisher's API provides means for businesses to register and manage their descriptions and offerings within UDDI registries by creating, modifying and deleting of the corresponding data structures.

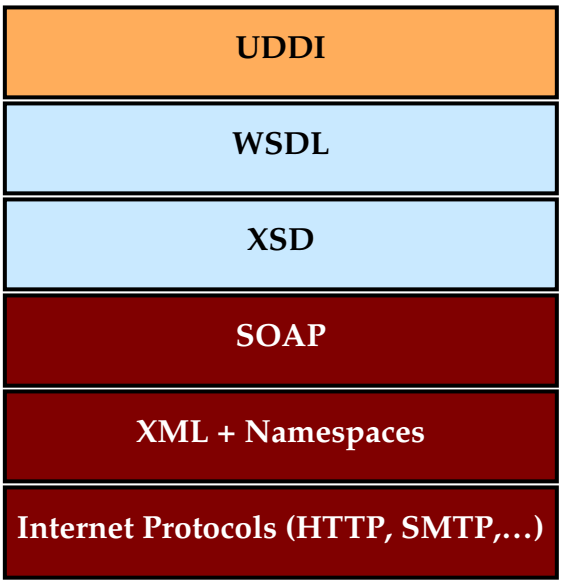

FIGURE 9 Web Services platform protocol stack

Thus, we have described the basic building blocks of basic SOA for Web Services. The corresponding protocol stack is graphically shown in Figure 9.

However, we remember from our discussion of SOC in Section 2.1 that basic SOA can be extended with some impressive advanced features and mechanisms for building value-added services. Certainly, these advanced mechanisms exist or are developed for Web Services as well, and represent serious technical challenges for researchers and developers of Web Services. The challenges include, but are not limited to, transaction management, security, composition, monitoring, billing, and contracting. We do not, however, intend to delve deeper into these features of value-added Web Services. The only challenge we will be directly looking into below in this Chapter is Web Services composition and workflow management. We will briefly review the major approaches for Web Service composition and survey the existing workflow languages for service composition in Section 2.4. But now we will concentrate on the concept of Semantic Web Services, which is of exceptional importance for understanding Web Services evolution in general, and the most promising of the Web Service composition approaches in particular.

2.3 Semantic Web Services

Despite recent significant advancements in terms of interoperability, current World Wide Web is still limited to the "browsing" paradigm of information exchange. In other words, it primarily gives human users a medium for consuming electronic information content. Humans have a natural capability of perceiving and interpreting information, the implicit meaning of which is not exposed for processing in a formal and explicit way. This type of capability is basically unavailable for automated systems performing information processing on the Web. Therefore, automation of various user activities on the Web, such as information retrieval and Web Service interoperation is desired [71]. In order to enable automated information processing on the Web such information must be reinforced with explicit and formal description of its meaning.

The Semantic Web is the evolution of the Web architecture, which has been envisioned and proposed to cope with the mentioned automation problem on the Web in particular. More specifically, the Semantic Web strives for the creation of mechanisms that augment content with *formal semantics*, thereby producing content suitable for the consumption of automated systems [14]. Some direct outcomes of such Web evolution are provision of automated functions and customization of information content for users. The Semantic Web, however, should not be seen as a separate Web but rather as an extension of its contemporary version, an extension in which information is enhanced with well-

defined description of its meaning better enabling computers and humans to interoperate.

Traditional Web uses XML-based markup for annotating information and links connecting documents and their parts. The Semantic Web extends this type of information description with the description of the meaning of such markup. Thus, each Web user may mark up sections of his/her personal Web page using arbitrary XML tags. These tags are often hard to interpret and almost impossible to subject to automated processing. The Semantic Web brings forward explicit description of the meaning of such markup by providing specific mechanisms for introduction, coordination, and sharing of formal data semantics as well as for reasoning about semantically described data. Resource Description Framework (RDF) [67]. This language, originally developed by W3C as a standard for specifying metadata on the Web, brings the description of abstract data model on top of a given XML markup in the Semantic Web. Not delving deeply into the details of RDF's organization, we merely note that it already provides the means for creation of machine-understandable information on the Web by describing *resources* (identified using URIs) using special *triple* constructs (object/attribute/value), which allow creating data models resembling well known knowledge representation model structures. Some partial understanding of data can be achieved and interoperable exchange of semantic information is made possible, especially when extending RDF with an object-oriented type system known as RDF Schema (RDF-S) [68]. Languages that build on top of RDF, such as DAML (DARPA Agent Markup Language), DAML+OIL [28], and OWL (Web Ontology Language) [70] utilize ontological approach to bring full-fledged formal semantics to data on the Web.

The ontological approach to formal data semantics description is based on the use of *ontologies*. Ontology (not to be confused with the abstract philosophical notion of ontology) is defined as a specification of a conceptualization [46]. More specifically, on the Semantic Web an ontology is usually seen as a document or a file that formally defines the relationships between terms in any particular domain of discourse. However, there are, in addition, upper ontologies (or foundation ontologies), which represent models of terms or objects applicable over a wide range of domain ontologies. An ontology typically comprises a *concept taxonomy* and a set of *inference rules* [14]. In other words, an ontology is a dictionary that describes the vocabulary used and provides means for semantic matching of syntactically different terms. Ontologies can be used to enhance the functioning of the Web in a variety of ways from a rather simple improvement of accuracy and relevance of Web searches' results to performing sophisticated reasoning procedures over complex relationships between various concepts and pieces of information. Ontologies play the central role in the Semantic Web, since they provide formal means to develop the desired capability of automated information processing. Ontologies are envisioned to enable the sharing of information

concepts, their reuse across multiple applications, the mapping of concepts between different ontologies, and the composition of new concepts from multiple ontologies [71].

Web Services, representing remote functionality invocable over the Internet, already provide a step towards automation of various activities on the Web. Semantic Web Services [71] is one more step further in this direction, an even more far-reaching perspective on Web automation complementing traditional Web Services.

In the initial vision by Berners-Lee *et al.* [14] Semantic Web Services are supposed to be primarily used not by human users, but by software agents, autonomous computer programs capable of accomplishing specific tasks on behalf of human users. To make use of Web Services, such agents need formal machine-processable descriptions of service capabilities and service interfaces. If an agent has a precise understanding of what a service does and how it can be accessed, the agent is then able to decide whether to use that service or not and finally invoke the service. Such precise understanding implies that the service is described comprehensively and unambiguously and with regard to the limited reasoning capabilities of software agents. Providing efficient means for creation of appropriate computer-interpretable service descriptions is one of the primary assignments of such Semantic Web languages as DAML and OWL.

DAML (and consequently its successor OWL) is a semantic markup language. It extends XML and RDF (RDF-S) to provide a set of constructs for creating machine-readable ontologies and markup information [76]. The essential part of the DAML program's Semantic Web contribution is Web Ontology Language for Services (OWL-S) [66]. OWL-S (formerly known as DAML-S) is an ontology of services, which gives providers of Web Services markup language structures for describing their Web Services in an unambiguous, machine-understandable format. OWL-S ontology consists of three main parts:

- *Service Profile* specifies the prerequisites and the effects of the service;
- *Service Model* reflects the operation of the service;
- *Service Grounding* describes the usage of the service in terms of its functional attributes.

Using such semantic description of Web Services the following significant tasks on the Web can be automated [71]:

- *Automatic service discovery* involving automatic location of services, which provide the necessary functionality and adhere to specific user requirements. This can be achieved by providing services with semantic markup describing their properties, which are related to services' classification and selection, e.g. provider's name, the intended use, payment methods, etc.

- *Automatic service invocation* referring to automatic execution of discovered services by software agents or other computer programs, which are capable of interpreting semantic markup describing the interface of the corresponding service function calls. Basic means for automatic execution of a Web Service are already provided with WSDL descriptions. However, they can be significantly enhanced with semantic descriptions, which provide functional and process metaphors of a Web Service service by thoroughly describing its inputs, outputs and operation semantics. In this way agents or other programs that invoke Web Services may automatically construct service requests and interpret service responses.
- *Automatic composition and interoperation* assumes the automatic construction of composition plans, which specify how to coordinate multiple Web Services to perform specific tasks that comply with given high-level descriptions of user-specific goals. Apart from utilization of means for automatic execution described above, automatic composition can be achieved by appropriately describing Web Services in terms of their preconditions and effects, as well as the tasks the services perform. Services can be automatically composed provided that the goals they want to achieve are interpretable, their effects are well understood and the prerequisites to their execution are satisfied.
- *Automatic execution monitoring* encoding the process of real-time tracking of the status of execution of user-specific requests. This is especially useful for controlling the execution of service compositions.

Service ontologies are particularly interesting from the viewpoint of this thesis because they already provide semantic means that can be used in mechanisms that perform service composition. First of all, OWL-S service profiles already distinguish between atomic and composite Web Services. Service models may include process models of the described services as a subclass. These process models describe composite services in terms of subprocesses along with inputs, outputs, preconditions, postconditions, and post-effects. Finally, such process models provide simple flow control constructs to specify the layout of the described composite process, and they orchestrate its components.

As a more general and far-reaching outcome, the Semantic Web allows to elevate the core mechanisms of the Web and Web Services to a semantic level, which leads to a more sophisticated description of functionality and to possibility of shared understanding of terms between different parties of service provisioning process via the exchange of ontologies that provide the necessary vocabulary for communication. This consequently makes autonomous agents perform "partial matches" [40] between service requests and advertisements and compose such partially matched services to obtain required functionality. Such approach scales beyond mere standardization and simple interface sharing and can be utilized on demand and in an ad hoc fashion, which brings information system

interoperability to a new, unprecedented, semantic level called *serendipitous interoperability* [61].

As we can see from the previous paragraph, the current understanding of the Semantic Web already differs in some aspects from the initial vision of it presented by Berners-Lee *et al.* [14] in 2001. The initial perception of the Semantic Web as a metaphor for a world-scale, universal, and distributed computational device has been revised lately. The traces of this revision process and of the Semantic Web evolution are well identified in the recent article by Nigel Shadbolt, Tim Berners-Lee and Wendy Hall [103]. They argue that the initial idea underlying the development of the Semantic Web [14] was rather straightforward and oversimplified. The vision of a fully automated agent-mediated Web appears a more distant perspective than when it was initially proclaimed, since in order to flourish agents need well established standards. This challenge has not however been abandoned – it has moved from the forefront to the more distant future. Another important revision of the Semantic Web paradigm concerns the initial aspiration to utilization of ontologies as conceptualizations of interrelated data on the Web. As the current Web is mainly consumed by humans, who establish their own particular views on and demands for information, a growing need for data integration is observed. In such a scenario it is extremely difficult to enforce the use of a single all-encompassing Ontology of Everything joining together an agreed set of particular domain ontologies and in such a manner creating a complete conceptualization of the world. Ontologies are currently seen mainly as mediation objects, which provide vocabularies for efficient sharing of meaning of specific data between parties interacting through the Web. In this ad hoc way ontologies promote serendipitous interoperability and information integration on the Web.

2.4 Web Service Composition

Service composition has emerged as a capability to be supported by service-oriented computing [84] systems and architectures. Such systems adopt services as fundamental platform- and network-independent elements described in machine-readable format when striving for ubiquitous and automated service discovery, communication, invocation and integration. Web Services are a typical and the most elaborate example of SOC paradigm realization. Web Services' native SOC capabilities, namely, description, discovery and communication are implemented using WSDL [26], UDDI [27], and SOAP [78] standards respectively. Service composition is used as the method for development of specific applications by reuse of other applications exposed for a public use as services.

Methodologically, service composition takes its roots from such scientific domains as Artificial Intelligence (AI), business process integration and workflow

management. Workflows were first used for modeling and implementing complex distributed business processes within enterprises. They can be viewed as plans of control flows reflecting temporal and causal relationships between component processes, and data flows between them. Such plans are formal and unambiguous representations of business process integration procedure. Composing Web services is essentially the same type of procedure as business process integration. So, the workflow-based languages and standards can often be successfully exploited for solving the Web Service composition problem.

Many industrial activities in service composition are devoted to the development of XML-based specification languages for specifying complex business processes and composite Web Services. They are often called Web Service Orchestration/Choreography languages and operate on top of native Web service capabilities UDDI, WSDL and SOAP. Here we survey only those languages which are actually used for Web service composition. First Web service composition languages XLANG [108] and WSFL [59] were proposed in 2000 by Microsoft and IBM respectively. They later joined their efforts to create the Business Process Execution Language for Web Services (BPEL4WS) [3], which finally resulted in the specification of its successor WS-BPEL [7] by the OASIS standardization body. In parallel to this evolution two more standards have been developed: Business Process Modeling Language (BPML) [6] proposed by Business Process Management Initiative, and Web Service Choreography Interface (WSCI) [5] jointly developed by Sun, SAP, BEA and Intalio, later submitted as W3C (World Wide Web Consortium) note, and finally resulting in the specification of Web Service Choreography Description Language (WS-CDL) [53] by W3C. Although all three branches of Web service composition standardization have their own merits and drawbacks, BPEL4WS is the standard that covers the most modeling aspects in a powerful and expressive enough way. Furthermore, it has received the most recognition and support from industry: numerous companies have pledged BPEL4WS support in their software products. A good detailed survey of Web service composition languages can be found in [117].

Although industrial Web service composition standards are quite expressive and capable of modeling specific business-related aspects of service composition, they often lack formality and precision. So, they might prove insufficient for robust automated composition of complex process models. On the other extreme there are strictly formal and precise modeling languages with a strong scientific background, which can be effectively used as the theoretical backbone for industrial languages. The most common examples of such languages are Petri nets [89, 94, 49, 79], π-calculus [77], and Finite State Machines surveyed in [76].

However, all the described languages allow only a manual service composition, i.e., a human composer has to formulate the composition goals, find the appropriate services, match them to the existing requirements and create a composition plan before the actual composition execution. Formal theoretical

languages such as Petri nets can only alleviate the efforts of the composer by providing means for automatic simulation, analysis and verification of the built plan. In order to develop an automated service composition framework, the tasks of service discovery, service matchmaking and control flow construction have to be performed without human intervention. Two major degrees of service composition automation are widely recognized: semi-automated and full-automated service composition.

Semi-automated service composition usually implies that service discovery and service matchmaking tasks are performed automatically, while control flow construction is done manually by a human modeler. Basic means for automation of service discovery process are provided by standardized globally available UDDI repositories [58]. However, automated service discovery still poses some serious challenges such as the huge number of service repositories, different languages, ontologies and business models. These problems are supposed to be successfully solved by the establishment of common and elaborate standards for semantic service descriptions [71] (some possible candidates for becoming the standards are described below). The scalability problem is especially important since searching for services through repositories of different size on the world-wide scale is an extremely difficult task.

Service matchmaking is much more difficult to automate, since current Web service description standards, WSDL and UDDI, lack formal semantics, i.e. it is impossible to capture the meaning of a Web Service according to its syntactic description. To bring semantics in these services' descriptions, Semantic Web languages such as OWL [70] and Web Service Modeling Language (WSML) [20] can be used in conjunction with service specification ontologies OWL-S [66] (or its predecessor DAML-S [4]) and Web Service Modeling Ontology (WSMO) [95], and with appropriate domain ontologies. Then the task of service matchmaking can be reduced to a logical reasoning task of finding a match between formal, logical expressions describing the requirements on service capability and a semantic service specification, which describes this capability using logical expressions [58]. A similar approach based on OWL and DAML-S has been proposed by Sirin et al. in [104]. They even developed a step-by-step control flow construction procedure, in which the semantic reasoner built on Prolog offers an appropriate service on every step of a composition plan construction to help the human modeler. Nevertheless, even such semi-automated approach cannot be widely applied for Web service composition, especially taking into account the unstable and dynamic nature of Web Services in many situations where the human factor poses a serious bottleneck.

Full-automated service composition ideally implies that the role of a human is limited to merely specifying a composition request. Such service request usually contains the specification of initial state, goal state and possibly optimization criteria [58]. Then the task of the automatic planner is to find the appropriate

candidate services, match them to established restrictions, build a plan or a set of plans, which transform the initial state into the goal state, and finally select a particular service enactment using a given optimization criteria. Formal aspects of composition requests are well described in [22]. Finding a path from the initial state to the goal state is a non-trivial task, which basically assumes utilization of AI planning techniques. Some of the AI planning techniques worth mentioning in the context of service composition are situation calculus, hierarchical task networks (HTN), rule-based planning and theorem proving. Narayanan and McIlraith [80] utilize the based on situation calculus to build a reasoner for automatic service compositions on top of DAML-S service descriptions. Sirin et al. [105] present a service composition framework based on HTN-planner SHOP2, which transcribes OWL-S services into HTN-tasks, which can be further decomposed into more primitive subtasks and linked by the planner into a composite HTN describing the final goal of the client. The drawback of this approach is that candidate services should be supplied to the planner as an input, which means that the human composer is still needed. Medjahed [72] proposes a service composition technique based on high-level declarative descriptions written in the specific description language CSSL (Composite Service Specification Language). To generate a valid composition plan, the method utilizes a set of syntactic and semantic composability rules that allow determining whether two services are composable or not. Another rule-based planning approach is SWORD [90], which adopts the Entity-Relation (ER) model to describe services and Horn-type rules to reason on top of services' preconditions and effects. Rao et al. [92] describe an automatic service composition method for Semantic Web services via Linear Logic theorem proving.

Recently it was realized that successful automation of the Web Service composition process requires formal explicit descriptions of composition goals, so that the goals can be automatically extracted, interpreted, and explicitly managed and reused. Goal-driven automatic planning of composite Web Services can be significantly enhanced by utilization of semantic goal descriptions and goal ontologies, such as GDL4WSAC [60].

Another interesting and modern challenge related to Web Service composition is dynamism [43]. Service composition dynamism combines two important characteristics of the composition process: the degree of automation (discussed above) and the time of services binding. Late binding, i.e. when component services are bound to a composition plan at run-time, not at design-time, would lead to remarkable improvement in composite services' flexibility and reliability.

A conceptual all-round solution to tackle the Web Service composition problem (among other Web Service related challenges) on a global scale is Web Service Modeling Framework (WSMF) [39]. This framework presents a really comprehensive and complex approach for bringing interoperability to Web

Services on the largest possible scale. The targeted interoperability is manifold: it includes mediation not only of data structures, but also of process models, business logic, message exchange protocols and even network protocols. The framework relies on the use of ontologies, goal repositories, Web Services and mediators and follows the principles of strong decoupling of all components and scalable mediation of all types of interactions within the environment.

The use of agents for automatic invocation, execution, composition and interoperation of semantically described services on the Web was envisioned and widely predicted with the emergence of the Semantic Web [14, 71, 51]. However, agent-based Web Service composition approaches stand slightly apart from the approaches described above. They are generally more graceful and flexible compared to traditional orchestration approaches, but they are directed to the future of the evolving Semantic Web. Agent-based approaches are based on the choreographic composition paradigm. As described in [36], agents treat Web Services as specific computational resources, which they can manage and trade to other agents in a multi-agent environment, and compose them in a proactive fashion via coordination with other agents representing other Web Services. Thus, agent-based service composition is conceptually and technologically different from the orchestration approach we follow in this thesis. That is why we do not intend a detailed survey of agent-based composition approaches here.

Some other interesting approaches and solutions to Web Service composition, which are worth paying attention to, are discussed in [11, 115] and extensively surveyed in [93, 57, 76, 58].

A newly emerged and rapidly evolving branch of Web Service Composition research is Context-aware Web Service Composition. Despite the fact that current number of contributions in this field is fairly modest, we have to mention some of them as they are relevant to our own work. The existing approaches either aim at context modeling for further utilization by/in Web Services, [16, 63] or implement Web Services and Web Service composition frameworks which utilize context in rather limited ways, e.g., [112]. An interesting technical solution by Keidl and Kemper [56] can be applied as the foundation for future Context-aware Web Services. It presents an extension to SOAP and SOAP-based messaging among Web Services to describe context at elementary level and successfully disseminate it throughout interoperating Web Services. The utilization of such messaging framework, as authors show, can provide a solid base for context-aware mechanisms for adaptation and composition of Web Services.

As a summary of the related service composition efforts, it should be stated that although the problem is given a broad recognition and a wide variety of research activities are currently devoted to solving it, full-automated service composition as one of the ultimate challenges of Semantic Web has still not been materialized in the form of mature approaches and solutions which could have wide application in current Web.

2.5 Petri Nets

Petri nets are a tool for the study of systems. Petri net theory allows a system to be modeled by a Petri net, a mathematical representation of the system [89]. Further, Petri nets allow not only modeling of systems, but also provide an efficient means for system analysis. The analysis of a Petri net model of a system can help determine important system properties and characteristics, and thoroughly study its behavior and structure. As a result, such analysis of the modeled system can be extremely beneficial for achieving a proper design of the real system and for avoiding possible errors and other inefficiencies during its operation. Thus, Petri nets are a formalism for system analysis, and its application is via modeling of systems and processes.

A model is basically an abstract representation of a real complex object. It is usually a mathematical or graphical representation, which describes the object or system with a certain level of abstraction. An abstraction created during modeling implies that some of the properties of the modeled object are important for the modeling purposes and others are not. Building a representation this way, modelers can significantly reduce the complexity of the model corresponding to the system under study while still getting a sufficiently accurate representation with respect to their goals. Study of a real system is often connected with certain risks, inconveniencies, excessive costs or even danger. It can be even impossible. That is why modeling is useful. The risk factors involved with real systems can be neglected; we obtain desirable system information by manipulating the representation instead. Therefore, modeling techniques are being successfully applied in numerous areas of human and natural sciences such as physics, biology, astronomy, sociology, economics and certainly computer science.

The development of Petri net theory started with the dissertation of Carl Adam Petri in 1962. The apparatus of Petri nets has changed significantly since then, but the main idea of the approach has remained unchanged. Petri nets were designed to model complex component systems and processes, interactions between the components and causal relationships between events occurring within them. Petri nets are currently a mature and thoroughly developed modeling tool. It has undergone a great number of modifications and enhancements in recent years. It is widely applied for solutions of various design and analysis problems in computer science and other fields. Among the strengths of Petri nets formalism is the pictorial graphical representation of built models, focus on interactions and dynamic relationships between systems' and processes' elements, capturing causality between events in complex processes.

Petri nets are practically in two ways. One approach implies that the design of a system is made using some other technique. Petri net model of the system is used for analysis purposes only, thus playing an auxiliary role. The built Petri net

model of the system is analyzed to discover any shortcomings or inefficiencies in the current system's design. Whenever such flaws are found during the analysis, the design of the system is rebuilt to overcome them. The analysis process continues its reiterating cycle until no more deficiencies are revealed. The system can be a real fully operational system as well as being under construction. This approach can be characterized as a constant refinement of system's design through external model analysis.

Alternatively, the system can have a Petri net model in the core of its design. So the whole system design and refinement process is made in terms of Petri nets. In this case the idea is not just to analyze the model, but to build the model, which is error-free, and further implement the model in the form of a working system using specific implementation techniques.

2.5.1 Petri Net Basic Formalism

The basic formalism of Petri nets represents the initial view of this modeling approach established by its creator Carl Adam Petri. Petri nets constructed using this basic formalism is often called ordinary Petri nets. This fundamental formalism includes a minimal set of allowed Petri net constructs and imposes some restrictions on the form of valid Petri net models. In contrast to the basic formalism there is a number of extensions and modifications to it, which either reduce Petri net models to describe more specific cases and processes, or extend the basic modeling instrumentation to increase the modeling power of the basic formalism remarkably. We will consider one of these extended formalisms in Section 2.5.2.

2.5.1.1 Formal Definitions

Usually, Petri nets are formally defined and manipulated in terms of sets (or *bags*, to be more specific) [89]. Set theory is especially useful when it is necessary to algebraically manipulate graphs and other structured graphical representations. An alternate way of defining Petri nets is via matrix algebra, which we do not pay much attention to in this work. Here, we give definitions of Petri nets and their elements in terms of set theory. Graph-based representation of Petri nets is discussed below.

According to [31], a Petri net is a tuple (S, T, F, M_0, W, K), where

- S is a set of *places*.
- T is a set of *transitions*.
- F is a set of arcs known as a *flow relation*. It is subject to the constraint that no arc connects two places or two transitions, or more formally: $F \subseteq (S \times T) \cup (T \times S)$.
- $M_0 : S \to \mathrm{N}$ known as an *initial marking*, where for each place $s \in S$, there are $n \in \mathrm{N}$ tokens.

- $W : F \to \mathrm{N}^+$ known as a set of *arc weights*, assigns to each arc $f \in F$ some $n \in \mathrm{N}^+$ denoting how many tokens are consumed from a place by a transition, or alternatively, how many tokens are produced by a transition and put into each place.
- $K : S \to \mathrm{N}^+$ known as *capacity restrictions*, assigns to each place $s \in S$ some positive number $n \in \mathrm{N}^+$ denoting the maximum number of tokens that can occupy that place.

This is commonly one of the most formal definitions for a Place/Transition type of Petri Nets. However, there exist a variety of other definitions, most of which skip arc weights and capacity restrictions. For simplicity we will also neglect these elements for now and use another less specific definition proposed by J. L. Peterson [89].

According to this definition a Petri net comprises four main elements: a set of places, a set of transitions, an *input* function, and an *output* function. The input function represents a mapping from a transition to a set of its *input places*, while the output function maps a transition to its *output places*. The former two elements define the structure of a Petri net, while the latter two – its connectivity.

A *Petri net structure*, *C*, is a four-tuple $C = (P, T, I, O)$; where

- $P = \{p_1, p_2, ..., p_n\}$ is a finite set of places, $n \geq 0$.
- $T = \{t_1, t_2, ..., t_m\}$ is a finite set of transitions, $m \geq 0$.
- The set of places and the set of transitions are disjoint, $P \cap T = \varnothing$.
- $I : T \to P^\infty$ is the input function, a mapping from transitions to bags of places.
- $O : T \to P^\infty$ is the output function; a mapping from transitions to bags of places.

The cardinality of sets *P* and *T* are *n* and *m* respectively. Thus, arbitrary elements of the sets *P* and *T* can be defined as $p_i, i = 1,...,n$ and $t_j, j = 1,...,m$ respectively.

Providing that places and transitions of a Petri net are defined by sets *P* and *T*, functions *I* and *O* are used to describe the connected Petri net.

Example 1. Let us consider a Petri net consisting of 4 places and 3 transitions. So, $P = \{p_1, p_2, p_3, p_4\}$ and $T = \{t_1, t_2, t_3\}$. A valid variant of the net connectivity can be described by the following values of input and output functions:

$I(t_1) = \{p_1, p_3\}$, $I(t_2) = \{p_2, p_3\}$, $I(t_3) = \{p_4, p_4\}$;

$O(t_1) = \{p_2\}$, $O(t_2) = \{p_1, p_3, p_4\}$, $O(t_3) = \{p_2\}$.

From this example several principles of building Petri nets can be seen:

1. Places can be connected to transitions and vice versa, but places cannot be connected to places, and transitions cannot be connected to transitions directly.

2. Arcs connecting places and transitions are always directed.
3. Every transition may have multiple input places and multiple output places, which is reflected by the fact that the values of input and output functions are sets.
4. One place can be simultaneously an input and an output place for the same transition, as in case of the place p_3 and the transition t_2 from the above example.
5. A place can be a multiple input or a multiple output of a transition, as in case of the place p_4 being the double input for the transition t_3. Although this feature was not originally allowed in basic Petri nets, it is quite commonly used. This leads us to the conclusion, that the values of input and output functions are in fact not sets, but *bags*, extension of sets that allows for multiple occurrences of the same element in a bag. Multiple inputs and outputs are often denoted by using arc weights.

2.5.1.2 Graphical Representation

Another way of defining and manipulating Petri nets additional to formal mathematical representation is the representation of Petri nets as graphs.

In terms of graph theory, a Petri net is a connected directed bipartite multigraph.

Petri net graphs have two types of nodes. One type corresponds to places and is usually pictured as a circle. The other one corresponds to transitions and is denoted with a (usually vertical) bar.

Nodes are all connected to each other with directed arcs pictured as arrows. Therefore, a Petri net is a connected and directed graph.

Multiple inputs and outputs of a transition can be presented as multiple incoming or outgoing arcs, or alternatively with weighted arcs. That is why a Petri net is a multigraph.

Finally, a Petri net is a bipartite graph because it has two separate sets of nodes, and all the arcs connect the nodes from different sets.

A *Petri net graph* G is a bipartite directed multigraph, $G = (V, A)$, where $V = \{v_1, v_2, ..., v_s\}$ is a set of vertices and $A = \{a_1, a_2, ..., a_r\}$ is a bag of directed arcs, $a_i = (v_j, v_k)$, with $v_j, v_k \in V$. The set V can be partitioned into two disjoint sets P and T such that $V = P \bigcup T, P \bigcap T = \varnothing$, and for each directed arc, $a_i \in A$, if $a_i = (v_j, v_k)$, then either $v_j \in P$ and $v_k \in T$ or $v_j \in T$ and $v_k \in P$ [89].

There is an exact correspondence between formal and graph representations of a Petri net. Figure 10 shows the Petri net graph built on the base of the formal definition from Example 1.

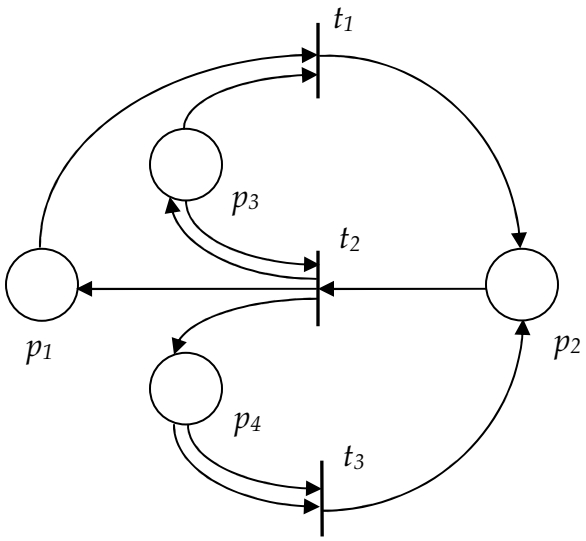

FIGURE10 Petri net graph from Example 1

2.5.1.3 Petri Net Markings and Execution

Since Petri nets are used to model dynamic systems and processes, they should provide some means of capturing dynamic changes. In basic Petri nets formalism no mechanism for structural alteration of Petri nets is provided. Petri net structure is supposed to remain unchanged. But there is a mechanism for describing the state of a Petri net, which can change during the simulation process. This mechanism utilizes another basic element of Petri nets – *tokens*.

Tokens can occupy places of a Petri net. They can be consumed by transitions from their input places through the input arcs and resumed to their output places through the output arcs. The primary role of tokens is to identify the state of a Petri net. A collection of all tokens residing in a Petri net at a moment of time over all the places of a Petri net is called a Petri net *marking*. A marking is the actual identifier of the state of a Petri net.

A marking μ of a Petri net $C = (P, T, I, O)$ is an n-vector $\mu = \{\mu_1, \mu_2, ..., \mu_n\}$, where $n = |P|$ and each $\mu_i \in N, i = 1,...,n$ is a nonnegative integer, which denotes the number of tokens in the place p_i.

The marking μ can be also defined as a function producing nonnegative integers and taking elements of the set P as arguments. The functional notation $\mu(p_i) = \mu_i$ is more general and hence more convenient in some cases.

Graphically tokens are represented by dots residing inside the circles representing the places. Positive integer numbers are usually used instead of dots if the number of tokens in a place exceeds 3.

The number of tokens assigned to one place can be practically quite large and is generally unbounded. This leads to the conclusion that the number of possible

markings of a single Petri net is infinite. All possible markings of a Petri net with n places comprise the set of all marking vectors, the dimensionality of which is equal to N^n.

Figure 11 shows an example of a marked Petri net with the Petri net structure pictured in Figure 10.

Now, when it is clear how the state of a Petri net is identified, another important question arises. How the state of a Petri net is changed? The process of sequential Petri net state changes is called the *execution* of a Petri net. The execution is performed by *firing* transitions of Petri net. Here, the main role is again played by tokens. However, this is more like a control function for them. Tokens act as prerequisites for firing transitions. A transition fires by removing certain number of tokens from its input places and depositing certain number (possibly different) of tokens to its output places.

A transition should be *enabled* before been fired. A transition is enabled if each of its input places contains at least as many tokens as the number of arcs connecting this particular place with the transition. Mathematically it can be stated that a transition $t_j \in T$ in a marked Petri net $C = (P, T, I, O)$ with marking μ is enabled if for all $p_i \in P$,

$$\mu(p_i) \geq \#(p_i, I(t_j)) \tag{2.1}$$

where $\#(p_i, I(t_j))$ is the number of entries of p_i in the input set of the transition t_j, i.e. the number of input arcs from the place p_i to the transition t_j.

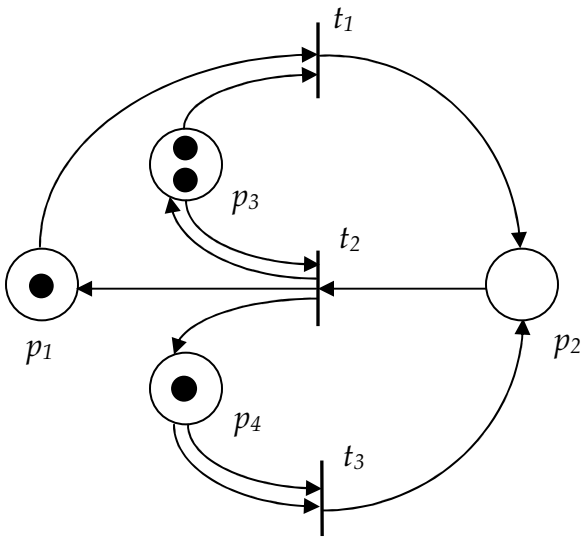

FIGURE 11 A marked Petri net

So, if the transition is enabled, on firing it removes the tokens from each of its input places and places tokens to each of its output places on the basis of the rule "one token per each arc".

For example, transition t_1 with $I(t_1) = \{p_1, p_3\}$ in Figure 11 is enabled, since both p_1 and p_3 contain tokens and are connected to t_1 with a single arc. If it fires, it takes one token from p_1 and one token from p_3 and puts one token to p_2 because a single arc connects p_2 and t_1.

Transition t_3 from Figure 11 is not enabled, since its only input place p_4 has only one token in it and two arcs connect this place to the transition. Therefore, the transition t_3 cannot fire at current moment and in current Petri net marking. Should the place p_4 have at least two tokens, the transition t_3 would become enabled and consequently would fire consuming two tokens from p_4 and putting one token to p_2.

The change of a Petri net state is reflected as the change of the Petri net marking. Generally, the marking μ transforms into the marking μ' on firing a transition in a Petri net.

If in a Petri net with a marking μ there exists an enabled transition t_j it may fire at any time resulting in a new marking μ', which can be defined as

$$\mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j)) \tag{2.2}$$

for all $p_i \in P, i = 1,..., N$.

The execution of a Petri net is a sequence of transition firings, which changes the markings of the Petri net. Transition firings can continue until no enabled transitions exist in the net. If at a certain marking there are no enabled transitions, the execution halts. At the other extreme, the execution of a Petri net may continue infinitely if there is a cycle in the sequence of reachable Petri net markings.

Let us consider the execution of the net shown in Figure 11. Initial marking $\mu^0 = \{1,0,2,1\}$. As it can be seen from the figure, at this marking the only enabled transition is t_1. On firing it the Petri net obtains a new marking $\mu^1 = \{0,1,1,1\}$. At this marking the only enabled transition is t_2 Firing this transition results in the marking $\mu^2 = \{1,0,1,2\}$.

At this point an ambiguous situation arises. Two transitions t_1 and t_3 are enabled. Both of them can possibly fire. However, according to the rules established by the Petri net formalism only one transition can fire at a time, either t_1 or t_3. In the most generic case it is the matter of chance which one will fire first. In our example illustrated by Figure 12 transition t_3 fired prior to transition t_1. This leads to one possible sequence of transition firings. Should t_1 fire first, there would be another sequence of transition firings and possibly different sequence of Petri net markings.

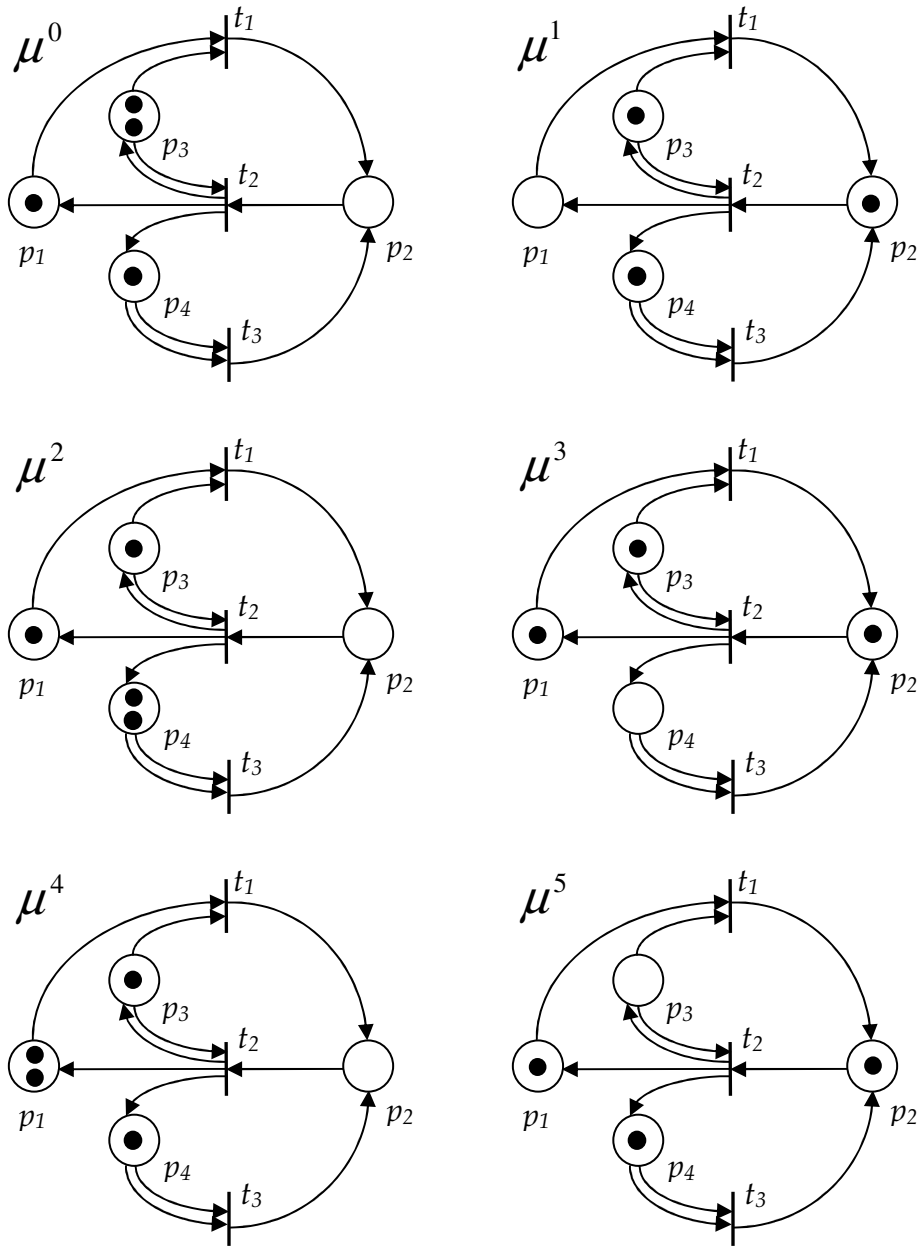

FIGURE 12 Execution of a Petri net

At marking $\mu^3 = \{1,1,1,0\}$ the situation is quite the same: both t_1 and t_2 are enabled and ready to fire. On firing t_2 a new marking $\mu^4 = \{2,0,1,1\}$ is reached. Now only transition t_1 is enabled. As it fires marking $\mu^5 = \{1,1,0,1\}$ is reached.

Marking μ^5 is a deadlock marking. Execution of the Petri net halts on reaching it because no transitions are enabled in this marking. By the way, this is not the only deadlock marking in this particular net. If other choices of transition

firings had been made during the execution, other deadlock markings would be reached.

2.5.1.4 Petri Net Classes and Extensions

Above we introduced in a formal yet general way the class of so-called ordinary Petri nets, which are often also called Place/Transition nets. This is a basic type of Petri nets, about which it is possible to say that it neither poses any specific restrictions on net structures, nor allows any extensions to the basic formalism. However, Petri nets formalism is a mature and elaborate modeling technique. Since it was invented it has been subject to a number of serious extensions, modifications and enhancement. The classification of the so far developed classes of Petri nets is huge. Though we do not have an intention to describe all the existing modifications even briefly, we will however mention here some of the interesting extensions of the basic formalism, some of which we plan to utilize in the forthcoming Chapters.

The well known restricted Petri nets classes are state machines, marked graphs, free-choice nets and simple nets. These formalisms mainly play with such immanent properties of Petri nets as *concurrency* and *conflict*, restricting or allowing either of them or both in some tricky manners. What is important about these classes of Petri nets is that they allow accurate and error-free modeling of some specific types of real-life processes.

Extensions of basic Petri nets formalism mainly strive for bringing additional modeling power to Petri nets to make them capable of modeling more sophisticated processes.

Basic extensions to ordinary Petri nets usually include merely new modeling constructs, such as *inhibitor arcs*, *disjunctive transitions, switch transitions*, etc., which basically change the rules of Petri net execution only in the locality of these constructs. We will show how these constructs can be used for increasing the modeling power of Petri nets in Chapters 3 and 4, where we will use some of them for addressing real modeling challenges.

Petri nets extensions, which modify the rules of execution globally, in entire nets, are usually treated as separate Petri net classes. Just to mention a few, some well-known classes of extended Petri nets are coloured Petri nets, time Petri nets, timed Petri nets, stochastic Petri nets and so on. We intend neither to describe these advanced formalisms here, since it would simply require lots of additional space and time, nor to utilize any of these extensions in our work described in the subsequent chapters. The only exception to this statement is the advanced formalism of Meta Petri nets, which we will directly apply in Chapter 4 due to our specific demands. This extension of Petri nets apparatus is considered in more detail in the next section.

One can see that in this section devoted to Petri nets modeling technique we have not touched upon the modeling and analysis aspects of Petri nets. We will cover these important issues to a certain extent in Chapter 3 devoted to modeling and analysis of Web Services using Petri nets formalism.

2.5.2 Meta Petri Nets

Meta Petri nets are multilevel Petri net structures consisting of two or more layers of similar Petri net structures. Multilevel Petri nets were proposed for facilitating flexible modeling and control of complicated dynamic processes in the work of Savolainen and Terziyan [99].

The main idea of Meta Petri nets is the following. A Meta Petri net consists of at least two separate levels: basic level and meta-level. The basic level contains a Petri net which simulates some complex process. The meta-level also contains a Petri net, and basically has no formal differences compared to the Petri net on the basic level. However, this metanet is used to simulate and help controlling the configuration changes at the basic level. Thus, the metanet is able not only to change the marking of the Petri net on the basic level, but also to reconfigure dynamically its structure. To achieve such an effect, in addition to two (or more separate) Petri nets residing on the separate levels, the Meta Petri net model includes a set of specific rules which create certain conformity between the nets on the two layers. The basic principle of such control rules is the following.

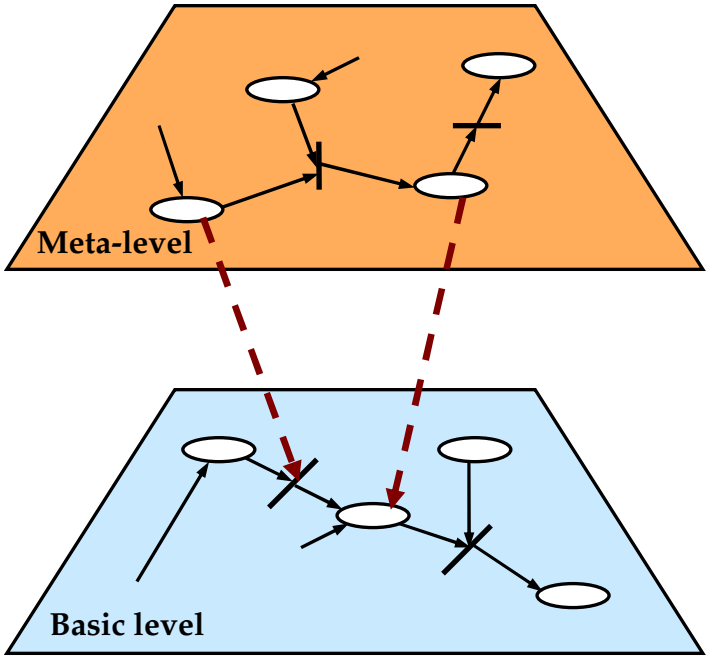

FIGURE 13 Basic principle of Meta Petri net organization

Certain places of the Petri net on the meta-level are associated with certain constructs of the Petri net on the basic level. Whenever one of these associated metaplaces becomes empty (i.e. stores no tokens), its corresponding structure on the basic level vanishes from the Petri net on the lower level. Conversely, if a token comes into a metaplace, its corresponding basic Petri net structure is restored on the basic level. By executing the Petri net on the meta-level tokens can be moved between metaplaces, which will result in a variety of Petri net configurations on the basic level.

A more general case of a Meta Petri net is multilevel Petri net, where the number of levels is conceptually unlimited. In such Petri net each higher level can establish the described configuration control over the lower layer. So, we obtain a sort of layered hierarchy, where lower layers are controlled and reconfigured by the higher ones.

According to [99] the major principles of Meta Petri nets' organization and functioning are as follows:

- the basic level of the model describes the domain in terms of objects, properties and relations;
- the meta-level describes rules of the dynamics in configuration modification in the lower level;
- the active structure of each level defines an active part pf the previous level at the current moment;
- similar tools are applied in each level of the model;
- the behavioral complexity of the modeled object defines the number of levels which are necessary for an exact description.

Formally speaking, the Petri nets of individual layers within the Meta Petri net can be defined as shown in Section 2.5.1.1. However, for a two-level Meta Petri net, we will call the places on the upper level *metaplaces* and the transitions on the upper level *metatransitions*. To formally build a two-level Meta Petri net, we, however, still need to define the linkage between the layers.

As it can be easily guessed based on the previous discussion, the linkage between the layers in a Meta Petri net is basically established by adding special meta-arcs, which originate from metaplaces and target either places or transitions on the basic level. Hence, there are two main types of metalinks: positional metalinks, which control the places on the basic level, and transitional metalinks, which control the transitions on the basic level. These links are illustrated in Figure 14 (a) and (b) respectively.

Accordingly, three major types of Meta Petri nets can be devised based on these two distinct types of linkage: positional Meta Petri nets, transitional Meta Petri nets and hybrid Meta Petri nets. We feel that the difference among these three types of Meta Petri nets is straightforward and is determined merely by the types of metalinks present in the net.

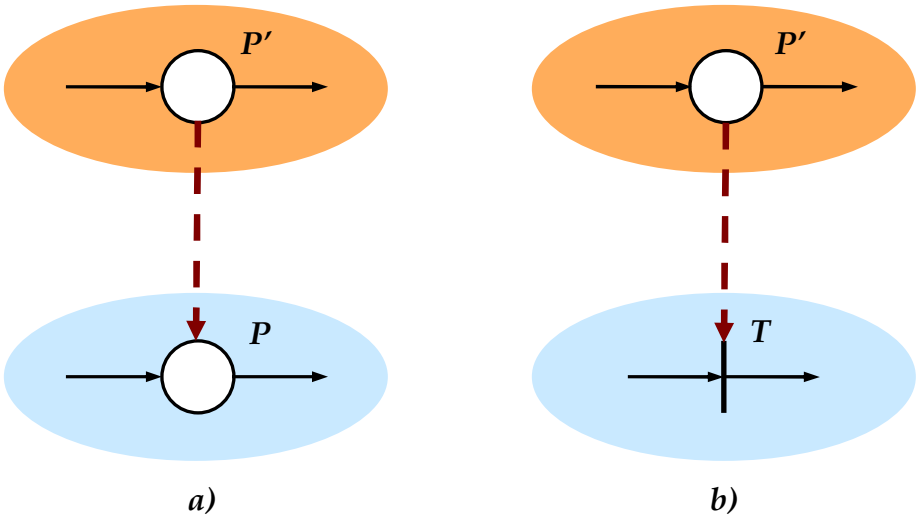

FIGURE 14 Two types of metalinks

Now we can define a Meta Petri net as the following tuple:

$$MN = (P, T, I, O, M_0, P', T', I', O', M'_0, R', Q', k') \tag{2.3}$$

where

- P,T,I,O and M_0 are elements of an ordinary Petri net;
- $P' = \{p'_1, p'_2, ..., p'_n\}$ is a finite non-empty set of metaplaces;
- $T' = \{t'_1, t'_2, ..., t'_m\}$ is a finite non-empty set of metatransitions;
- I' is the input incidence function mapping metaplaces to metatransitions;
- O' is the output incidence function mapping metatransitions to metaplaces;
- M'_0 is an initial marking of metaplaces;
- R' is the incidence function mapping metaplaces to places;
- Q' is the incidence function mapping metaplaces to transitions;
- k' is the optional parameter in a timed Petri net reflecting the number of time quanta between two consecutive firings of metatransitions, where one time quantum is set to the time period between two consecutive firings of ordinary transitions.

In Chapter 4 we will show how the Meta Petri net formalism can be applied for a successful solution to dynamic reconfigurability problems.

2.6 Context-Aware Computing

This section discusses Context-aware Computing paradigm and specifically the concepts of context and context awareness. It also highlights those major research achievements which we use later on as the foundation for the ideas developed in Chapter 4.

2.6.1 What is Context?

The notion of a "context" is somewhat vague to define. According to Merriam-Webster's Dictionary [73] *context* is "the interrelated conditions in which something exists or occurs". However, this definition is too general to be applicable for concrete types of problems. Therefore, a variety of specific definitions of the term "context" exists in different fields of science.

Perhaps the clearest example of the natural context usage is human communication. When people communicate they convey information, impressions and ideas to each other in a quite brief and concise form. What is more, they are capable of easily and properly capturing the conveyed information and appropriately reacting on that as a consequence. During a discourse people rarely encounter problems of misunderstanding that require disambiguation of misinterpreted concepts. This is not only because of the richness of the language they share and of the presence of common sense knowledge that allows for correct comprehension and subconscious reasoning. These two extremely important aspects of human communication are yet not capable of filtering all possible ambiguities arising during a discourse. The last but not least feature of human communication is human implicit understanding of the situation in which the discourse takes place including the subject of the discourse. In other words, "when humans talk with humans, they are able to use implicit situational information, or *context*, to increase the conversational bandwidth" [33].

The awareness of context is what lets humans to ignore occasional ambiguities in discourse and still interpret the received information correctly. Context often dictates such an interpretation of the information that it would be completely incorrect under different circumstances. This observation can be generalized to the statement that everything is true, correct or valid only in certain context. Only absolute things, which are hardly imaginable, and may not even exist, can be true or false regardless the context, or better to say, in all contexts. In Artificial Intelligence, this observation is referred to as the problem of generality, essentially meaning that any logic axiom, which can be constructed, holds only in a certain context, and it is always possible to devise a more general context, in which this axiom no longer holds [12, 69].

In computer science, and particularly in human-computer interaction context utilization would be of great use. Computers, and more specifically applications and services treat information in an absolute way and are not currently capable of processing context to improve their effectiveness, flexibility and usability. By supplying context to applications and services involved in a human-computer dialogue, new value-added services and applications can be created and the user experience can be significantly enhanced.

In order to use context effectively to solve concrete types of problems, it should be given an appropriate definition that particularly suits into the needs of that certain problem. Such specific definition basically determines what is considered the context, and what is not, and perhaps how context can be used.

Computer science researchers working on context formalization have given a variety of different definitions of context. A survey of them can be found in [23]. The common weakness of the majority of those definitions is trying to enumerate examples of what is context, instead of giving a comprehensive specification of context conceptual properties. For example, Schilit *et al.* [100] define context as location, identities of nearby people and objects, and changes to those objects. They further classify context into computing context, user context, physical context, and time context. Although descriptions of these mentioned context categories bring some clarification to the initial definition, the definition is still difficult to apply.

In contrast, there exist a couple of formal definitions of context. One is given by Chen and Kotz [23] and is as follows:

> *Context is the set of environmental states and settings that either determines an application's behavior or in which an application event occurs and is interesting to the user.*

Another formal definition is presented by Dey [33]:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.*

Though it is possible to say that the above definitions are loose compared to the example-based definitions, they are precise and comprehensive in the sense that they neatly determine the bounds beyond which the information cannot be classified as context. The enumeration of valid context examples may differ from one concrete application scenario to another, which is in fact a more plausible case than a fixed and invariant set of relevant contexts.

In spite of the apparent context usefulness for enhancing human-computer interaction and increasing applications' flexibility, context is not as simple to derive and properly interpret. So, in general it should be used with caution, and avoiding heavy reliance on its usage. The reason for that is that contextual information possesses a set of interesting properties, which can often cause serious problems for proper interpretation and verification of context.

It is straightforward from the above definitions that context is a specific representation of the world, or its part, or of a certain state of affairs. The specificity of such representation is the result of the following context properties or *context dimensions* described in detail in [12]:

- *Partiality*. The context always takes into account only a certain portion of the world, or a subset of a more comprehensive state of affairs.
- *Approximation*. The context always uses a certain level of detail for the description of a portion of the world, i.e. it abstracts away some aspects of the described state of affairs.
- *Perspective*. The context always entails the observation of the world from a certain point of view.

Besides the described context properties, which are immanent to context and hence permanently persist, there are a number of properties or features which may or may not inhere in context depending on the actual situation. Still, these characteristics, more thoroughly described in [52], can be vitally important when manipulating and interpreting context.

- *Volatility*. The context is not generally static. It tends to change at a sometimes varying level of dynamicity.
- *Imperfection*. The context can be *incorrect* if it fails to reflect the true state of affairs. It can be *inconsistent* if it combines contradictory information. Finally, it can be *incomplete* if some of the important aspects of the modeled state of affairs are unknown.
- *Multiplicity*. Semantically the same contextual information may receive multiple alternative representations in different syntactical forms, at different levels of abstraction and from different perspectives.
- *Interrelation*. Different pieces of contextual information can be related to each other directly or indirectly through other contexts, so that they form complex interrelations and sophisticated correlation patterns which are difficult to discover and analyze.

Concluding this discussion we give our own, specific and explicit definition of context. We do not however want to reinvent the terms. That is why we generally follow the definition of context presented by Dey in [33]. We modify this definition by merely making it more concrete to suit our specific purposes, which will become clear in Chapter 4. Thus, we define context as follows:

Definition 2.1. *Context is any information that can be used to characterize the situation of an entity, which affects the interaction between a user and a service (service application). The entities include the user, a user-specific task, the service (service application), a computing environment in which the interaction happens, and a part of the physical world involved or somehow related to the interaction.*

2.6.2 Context Awareness

Having defined the concept of context, we still need to define the usage of context. The usage of context is through *context awareness*. Apparently, context awareness is the property or capability of a computational entity to make use of context. Since 1994, when Schilit *et al.* [100] made the first effort on defining context awareness and context-aware computing, there have been numerous attempts to give appropriate definitions of context awareness. However, the majority of them have been too specific.

Perhaps, the most general definition of context awareness has been given by Barkhuus and Dey [10] and refers to context awareness as to "an application's ability to detect and react to environmental variables", i.e. context awareness is simply the ability to react to the (change of) context.

Another definition, which is ably balancing between being too specific and being too general, has been presented by Dey [33]:

> *A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*

We believe that this definition's strength is in giving accurate and comprehensive, while still general criterion for distinguishing between context-aware entities and those entities unaware of context.

A context-aware system can be generally represented using a "black box" representation. In order to demonstrate the difference between context-aware and context independent systems, let us first illustrate the traditional view on computer systems using the "black box" representation. That is, any system can be represented as an implicit structure, called "black box", the content of which is unknown. However, each such structure must have an explicit input and an explicit output. The function of the system is then to transfer its explicit input into its explicit output. The corresponding structure is presented in Figure 15.

FIGURE 15 The "black box" representation of the traditional computer system

A context independent system generally operates in a deterministic way (unless it is a random number generator or probabilistic functioning is the intent). It is supposed to always produce the same output from a constant input. Needless to say, this situation is not always acceptable.

There is also another type of traditional computer system, which can be called *interactive* system. This type is more interesting. It implies that the system facilitates somehow the participation of a user in its functioning, i.e. the system interacts with the user to achieve certain goals. Structurally, such a system also has an explicit input and explicit output just as in the previous case. But in addition it has a user-controlled feedback, which allows achieving appropriate results by manual attenuation of the system input. So, it is basically the role of the user to devise an appropriate input on the basis of the previously produced outputs. The corresponding system structure is illustrated in Figure 16.

The drawback with traditional systems is that they cannot generally perform their tasks without external, mostly human supervision. Explicit inputs and outputs can be characterized as slow and intrusive. Furthermore, they require special attention from the user especially in the sense of accurate defining of input values. As for the case of interactive loops, in which users are unconditionally involved, the sequential input-output looping seems neither increasing speed nor adding efficiency to the system's operation.

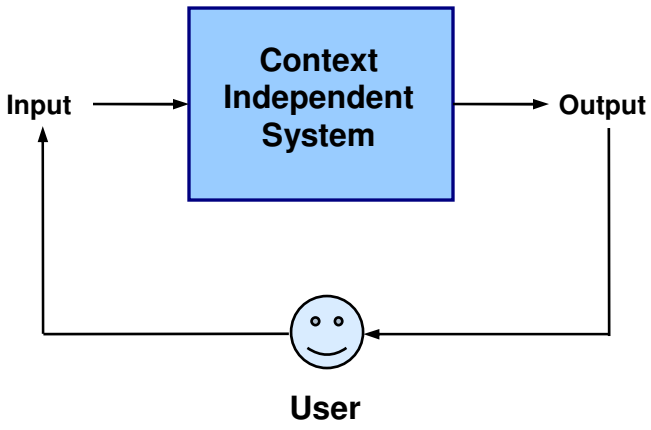

FIGURE 16 Interactive computer system

Context-aware computing paradigm solves, to a large extent, the problems indicated above. Introduction of context to a system as an implicit input instead of explicit interaction with the user achieves several important positive results. Implicitness of the contextual input means that it can be processed automatically and is basically transparent to the user. This leads to a conclusion that user intervention is no longer needed, and context can be processed faster and much more efficiently, thus significantly reducing explicit interaction. Secondly, a human user is out of the loop, since his intervention is no more necessary for obtaining desired results. Finally, the functioning of the system is no longer deterministic, because, after adding a new context input, different outputs can be obtained from the same explicit input. Moreover, within the presence of context the system becomes generally more flexible and responsive. So, context steps forward as a

new dimension, along which the operation of systems varies. The corresponding "black box" representation for a context-aware system is shown in Figure 17.

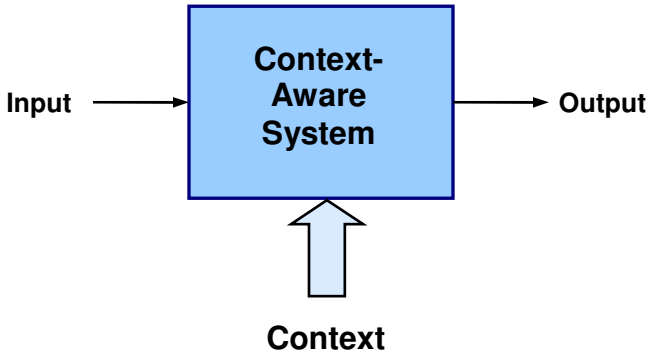

FIGURE 17 Context-aware system.

Along with the structural aspects of context-aware systems we just considered, there are issues related to functional applications of context, i.e. how exactly context can be used to enhance performance and flexibility of computer systems.

Context-aware applications are categorized by different researchers along different sets of so-called context-aware features identifying exclusive types of context uses. Schilit *et al.* [100] single out the following categories of context-aware applications:

- *Proximate selection*. This is basically a user-interface technique which highlights, emphasizes or makes somehow easier to choose objects located nearby.
- *Automatic contextual reconfiguration*. Restructuring of the system by adding, removing and substituting its components and by modifying the relationships between existing components with respect to context and its changes.
- *Contextual information and commands*. Instructions which can produce different effects with respect to the context in which they are issued.
- *Context-triggered actions*. Context-based rules specifying the adaptation actions to be taken in concrete situations.

The above categorization identifies the enumeration of specific uses of context rather than a comprehensive set of exclusive ways of context utilization. Pascoe [86] proposed a taxonomy of context-aware features, including contextual sensing, contextual adaptation, contextual resource discovery and contextual augmentation (association of digital data with the user's context). Finally, taking into account the above described categorizations, Dey [33] presented three main categories of contextual features:

- *Presentation* of information and services to a user.

- Automatic *execution* of a service for a user.
- *Tagging* of context to information to support later retrieval.

With regard to the presented categorizations two essential forms of context utilization can be distinguished. One is automatic adaptation of application's behavior with respect to observed context, and the other one is presenting the context to the user either on the fly or by storing it for later retrieval. According to Chen and Kotz [23] these two distinct forms of context usage force the classification of context into active and passive types. *Active context* then refers to the data that directly influence application's behavior, while *passive context* is the context which is relevant to the application, but is not critical to its operation. Based on such classification Chen and Kotz [23] give separate definitions to these two distinct types of context-awareness:

- *Active context awareness*. An application automatically adapts to discovered context, by changing the application's behavior.
- *Passive context awareness*. An application presents the new or updated context to the interested user or makes the context persistent for the user to retrieve it later.

Barkhuus and Dey [10] slightly expand the definition given above. While they define active context awareness similarly, they extend the concept of passive context awareness with an important aspect declaring that passive context awareness not only presents the context to the user but also lets the user decide how the application's behavior should be changed according to the discovered context. Moreover, they argue that there is one more type of context awareness, or a level of interactivity in their terminology, called *personalization*. "Personalization is where applications let the user specify his own settings for how the application should behave in a given situation" [10]. So, personalization is, perhaps, the most primitive, though the most common and elaborate type of context awareness. Personalization is not real-time context awareness: it implies that relevant user data and settings are supplied to the application prior to its actual operation. Personalization, often also called customization or tailoring, is a widely researched and utilized feature in modern computing. Many examples of its application can be discovered in our daily lives: from ringing profile settings in mobile phones to sophisticated questionnaire forms and personal profiles when visiting various web sites.

To differentiate somehow between the presented types of context awareness, an example is needed. A good example of a context-aware application is managing of ringing profiles within a mobile phone. Almost any mobile phone provides the possibility for a user to manually set one of the available ringing profiles. This type of functionality can be classified as personalization. If the phone is capable of detecting the change in the neighborhood, e.g. that the user has entered a meeting room, it may either change the ringing profile automatically or prompt the user to

confirm the change of the profile. The former option corresponds to active context awareness, whereas the latter one refers to passive context awareness capability.

Obviously, the types of context awareness are characterized by levels of explicit interaction between the application and the user. Personalization exhibits the most explicit interactivity level when most of the context is provided explicitly by the user. Passive context awareness implies that the level of explicit interaction with the user is moderate to low, i.e. merely critical contexts are negotiated with the user. Finally, in case of active context awareness the user should not intervene in the operation of the application.

Explicit interaction basically distracts the user, who is expecting that everything will be done without his/her participation. So, from this viewpoint active context awareness seems an ideal solution. However, there is one more interesting observation. Applications are generally not yet as intelligent as humans, which means that they are not always capable of devising decisions which would satisfy their human users. Such imperfection in applications can be explained by imperfection of context. While a human is generally able to properly interpret incomplete or inconsistent context, an application often fails to do that. As a consequence, it will act in a way different from the way the user would act in that situation. For example, when the user enters a meeting room and the application detects this, it may automatically switch the phone into a silent mode not knowing that the arranged meeting is informal and hence does not require switching off the ring tone. As a result, the user might miss an important call and would be very dissatisfied.

Using the black box representation, different types of context awareness can be illustrated and the distinctions between them can be clearly seen. Personalized context-aware system is shown in Figure 18.

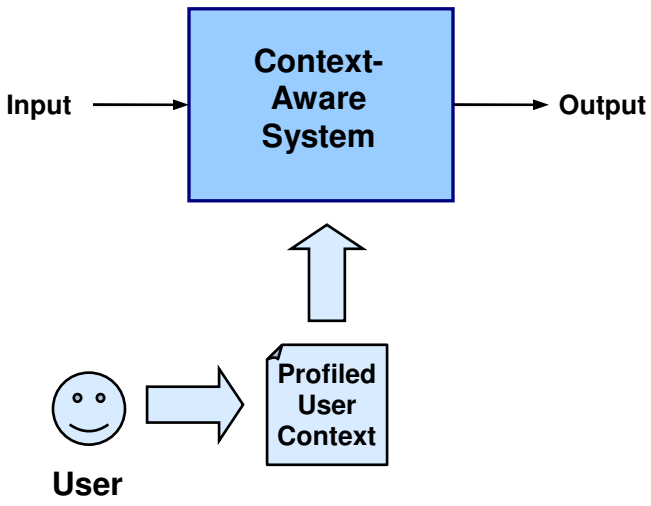

FIGURE 18 Personalization of a computer system

As can be seen from Figure 18, in case of personalization the interaction between the user and the system has some distinct properties. At first, it is unidirectional: the user explicitly interacts with the system by supplying his specific settings to a certain profile, but the system does not contact the user – it only takes the profiled context specified by the user. Such interaction scheme results in another specific property of personalization – the interaction is not direct but mediated by a certain profile storage. Finally, the interaction is asynchronous, i.e. the user may change the profile and the system may read it at arbitrary moments of time.

As we already mentioned, active context awareness does not imply any explicit interaction. Instead, it is supposed that active context-aware system or application operates solely on the base of the sensed context, which it is capable to acquire from the environment and which generally has no direct connection with the user. In such case the context-aware system usually senses the necessary context or infers it from other contexts, some of which can be made known directly by the user.

Figure 19 illustrates the operation of active context-aware system. The main distinction of such a system is that it is capable of performing automatic real-time self-adaptation. In case of personalization the system does not adapt itself during operation, but is adapted with respect to user context prior to actual operation. As for the case of passive context awareness, which we consider below, the system does not at all adapt itself, but gets adapted by the user.

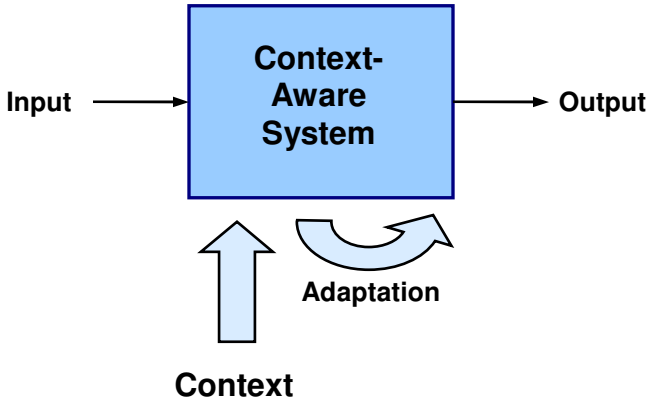

FIGURE 19 A system exhibiting active context awareness

Passive context awareness does imply explicit interaction with the user. At the same time it implies adaptation of the system according to the sensed context. Combination of these two aspects leads to an interesting concept of *user aware adaptation*, which we previously studied in [120]. User aware adaptation basically means that a user is responsible for making adaptation decision, though the system detects the necessity of making adaptation and provides the user with the relevant

contextual information. The user's adaptation decision is often called a *corrective action* [98]. The corresponding black box model is presented in Figure 20.

The distinction of the passive context-aware system is that it evolves in the explicit, direct, and real-time interaction with the user. Every time the system discovers the context in which it might need to adapt, it explicitly contacts the user and addresses the adaptation decision to him/her. It should be noticed that the adaptation decision is basically inferred by the system. However, the user is the one who takes responsibility to approve it or to reject.

Although active context awareness seems preferable to the passive one due to the autonomy of its operation, there are many cases in which passive context awareness can be beneficial. These cases are mostly connected with difficulties during reasoning procedure and can be roughly split into two major categories.

First category can be referred to as reasoning under imperfect context. Passive context awareness is generally applicable in cases like this when the system fails to confidently and unambiguously decide which actions are needed to conform with the current context and which are not. Imperfection of discovered context may often lead to the situation where the system cannot guarantee that the inferred adaptation decision will produce positive results to the final user. In such a situation redirection of the adaptation decision to the user is more than justified since one's own wrong decision generally distracts much less than the same decision made by someone else.

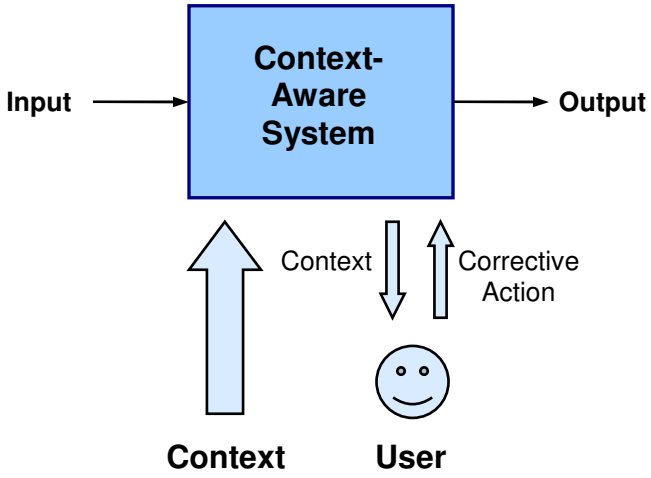

FIGURE 20 A system exhibiting passive context awareness

Another application category for passive context awareness is more straightforward. It can be referred to as reasoning failure. It might happen that the current context entails a conflict with the goals or demands established by the user, which means that the system cannot infer any acceptable adaptation decision under the current context to ensure a proper operation. The least what the

deadlocked system can do is to notify the user about the problems it encounters. Nevertheless, the system often can do more than that: it can propose a corrective action for the user to commit in order to ensure the operability of the system. The corrective action can be generally of two types: it aims at either revision of the user's demands or change of the current context. The example of the first type of corrective action can be a situation when the user wants to take a flight but it appears that it is currently a non-flying weather. In this case the corresponding travel planning service encounters a conflict which it can resolve by proposing the user to take a railway trip instead of a flight. So, the proposed corrective action requires the user to change his/her own preferences concerning the given task. Continuing this example, the corrective action of the second type would be, for instance, proposing the user to wait until it is again a good flying weather. Such corrective action requires the user to take certain actions in order to change context so that in a changed context the service can successfully complete the given task in its initial setting.

Thus, the goal of the system is to conform to the requirements, restrictions and preferences established by the user and at the same time adjust its behavior according to the changing context. Such problem can be basically reduced to a formal optimization problem of finding optimal adaptation decision. Though it is not our goal to present such formalization, we can illustrate the general principle of finding adaptation decision. Context can be generally represented as a subspace within an n-dimensional space of different contextual factors. The same can be stated about the user's restrictions. Then the area of possible adaptation decisions can be defined in such a space as an intersection of current context and user restrictions subspaces. This representation is illustrated in Figure 21, where context is represented by a plane for the sake of pictorial simplicity, and the area of optimal adaptation decisions is a part of the plane embraced by the red dashed circle and obtained as the intersection of the context plane and the user restrictions space.

We show the above representation intentionally. It helps us more clearly understand the idea of corrective action. It can be said that a corrective action is needed whenever current context space and user restrictions space do not intersect. This means that the context-aware system is not able to find a suitable adaptation decision. The corrective action aims at obtaining such intersection by modifying the mentioned spaces. The corrective action of the first type modifies the user restrictions space, while the corrective action of the second type modifies the current context space. As the result of applying the corrective action two spaces intersect, i.e. in the n-dimensional problem space there appears a finite or infinite subspace of adaptation decisions, which satisfy both the user's requirements and the current context.

As in the case with the definition of context, we conclude this section with the definition of context awareness which suits to the specific purposes of our work

described in Chapter 4. This time our definition is again based on the definition provided by Dey in [33] and is the following:

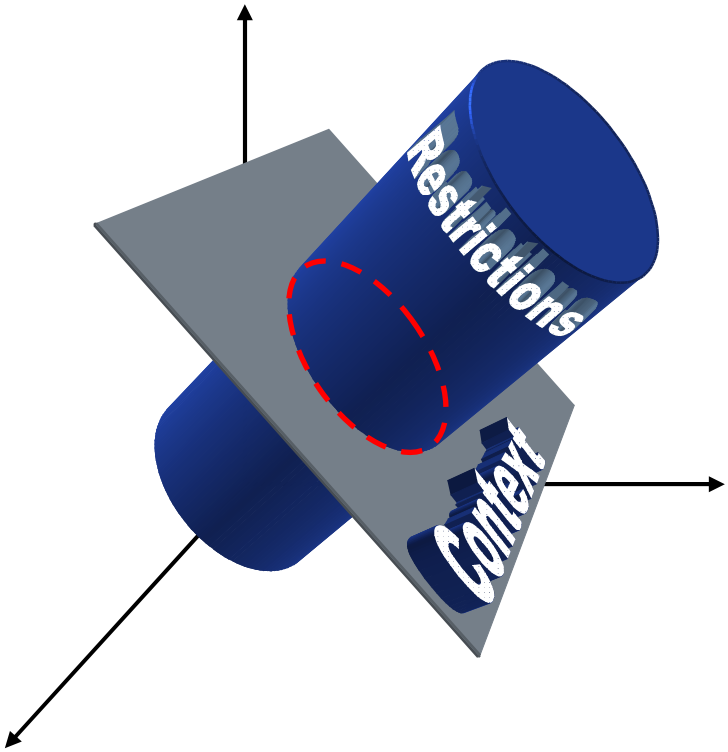

FIGURE 21 Graphical representation of context-aware adaptation problem

Definition 2.2. *A system is context-aware if it uses context to provide relevant services to the user, where relevancy depends on the user's task.*

Additionally we provide here our specific definition of a context-aware service since services and systems are generally not the same. So, when defining a context-aware Web Service in Chapter 4 we will implicitly proceed from the following specific definition of context-aware service:

Definition 2.3. *A service is context-aware if it uses context to change the structure of its execution flow and in such a way provides a relevant functionality to the user, where relevancy depends on the user's task.*

2.6.3 Context-Aware vs. Context-Based Services

One of the important issues to be discussed is that we have to make a distinction between context-aware and context-based services. There is a conceptual difference between these two types of context-sensitive services, which is to be investigated in this subsection.

Although we already defined the notion of context-aware system and context-aware service, it should be emphasized that a context-aware service makes use of a certain context by taking it as an additional input. This input may or may not be used by the service during its operation. The main idea of such an input is that it is used on demand or when necessary. So, for a context-aware service context inputs play auxiliary role.

In case of *context-based* services context inputs are in fact normal inputs, i.e., context is treated as an input parameter before starting service execution. Then the context is as necessary for the service operation as the other input parameters, without which the service cannot basically operate. Context-based services use context in a different way. While context-aware services utilize contexts in an indirect fashion, using them as a sort of control facility that help them change the structure of their execution flow (which we will be calling "behavior" for the sake of simplicity, assuming a somewhat restricted meaning of the commonly used word "behavior") when needed, context-based services treat context as normal input data that they directly use in their computation process. So, for context-based services context inputs are mandatory.

Figures 22 and 23 illustrate the usage of context by context-aware and context-based service respectively.

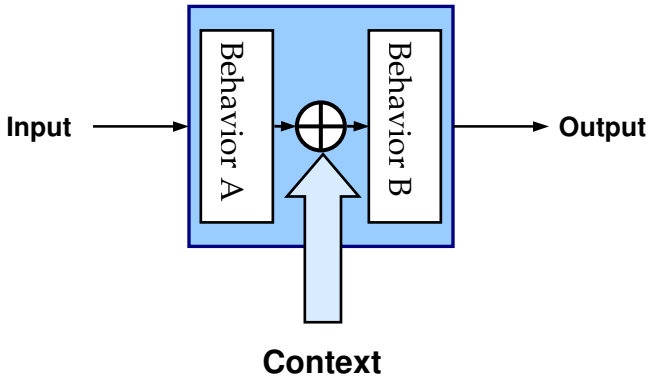

FIGURE 22 Context-aware service

Thus, the major distinction between context-aware and context-based service is that the former can operate when context is unavailable, while the latter cannot.

Perhaps the most common example of a context-based service is a location-based service. Location-based services typically take the current user location as an input parameter and perform certain proximity search with respect to that location. For example, they may find the nearest pizzeria or hotel.

Recently a serious discussion has been raised arguing the relevancy of this type of services to the class of context-sensitive services. Indeed, it can be seen even from the Figure 23 that such services do not make any conceptual distinction between context and other input data (except for the method of their acquisition).

Context-based services treat context equally to normal data performing their calculations and other operations based on it in a deterministic manner. So, context stops playing the role of external influencing factor, i.e., simply stops being the context. Fortunately, we are not interested in considering this type of services in our work, so they remain beyond the scope of this thesis.

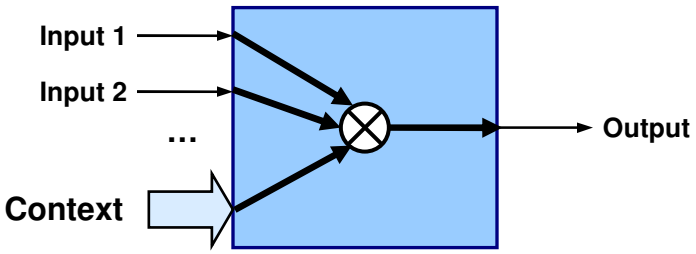

FIGURE 23 Context-based service

However, even in the absence of context-based services context-aware services cannot be generally brought to the other extreme described in Figure 22. The case shown in that figure implies that context can only be used as an external control triggering changes of service behavior. Nevertheless, it is still possible that context can be treated as both control and data. In this case the behavior of the service is changed also in the way that it takes the context as a data input for its operation. It should be noted that such a service would still be capable of operating without the context, since it utilizes the context as input data only after it has changed its behavior, i.e., only when the required context is already available and has effected the corresponding behavioral change.

2.7 Concluding Remarks

So far, in Chapter 2 we presented an overview of several modern and not-so-modern technologies and techniques related to the problem of Context-aware Web Service Composition in general and to our specific solution methodology of this problem in particular. The information provided here constitutes the necessary background and the set of relevant knowledge to allow a non-specialist reader to prepare himself/herself for the reading of the consequent Chapters and to ensure a smooth comprehension of the ideas and apparatus presented in them. Although it may seem that the concepts discussed in this Chapter comprise but a disjoint collection of independent research and technology areas, we will show during the next two Chapters how this concrete set of technologies and methodologies can be successfully combined to obtain an innovative and comprehensive solution to the problem of context-aware automated Web Service composition.

However, we briefly sketch our solution here to see how it builds upon the described paradigms and technologies.

Our solution to the problem of Context-aware Web Service Composition is a formal framework that provides a set of methods for modeling Web Services in a context-aware fashion. The apparatus of Petri nets will be used as the main instrument for formal graphical process-oriented modeling of Web Services. In the development of our Petri net based Web Service composition technique we will rely on the achievements in the domain of workflow modeling and process-oriented Web Service composition described in [49, 1, and 2]. In Chapter 3 we will develop and append, to our Petri net composition models, a declarative Web Service algebra similar to that described in [49].

In Chapter 4 we will show how contextual information can be utilized to enhance the process of Semantic Web Service composition. Although we will not directly make use of the semantics and semantic service description, we will highlight their exceptional importance as a prerequisite for successful application of the ideas developed throughout Chapter 4. In addition, we will outline some ways of how contextual information can be integrated with semantic descriptions and ontologies.

We will modify the idea of Hierarchical Task Networks AI planning technique to facilitate on-demand, goal-driven and automated Web Service composition. We will not touch the technological aspects of enhanced Web Service composition process, such as languages for composition goals representation and principles of semantic composition reasoner's functioning and organization. Nevertheless, we will present some conceptual enhancements to the procedures constituting the composition process. In particular, we will show how context can be utilized to increase the relevance of formal goal descriptions and to enhance descriptions of Web Services.

We will utilize the modeling framework developed in Chapter 3 to describe the procedure of automatic Web Service control flow construction on the basis of given composition goal descriptions. It will also shown that context can be a powerful means for increasing the relevance of composed Web Services already at the stage of service matchmaking and for reducing the complexity of composite Web Services process models.

Finally, we will develop a formal approach for modeling of context-aware composite Web Services and run-time reconfiguration of their structure with respect to context. This reconfiguration framework will be based on the application of the Meta Petri nets modeling formalism also described in this Chapter.

3 PETRI NETS FOR WEB SERVICE COMPOSITION

In this contribution Chapter we develop a specific representation of Web Services as process-oriented models using the Petri nets formalism. In this work we rely on the ideas and achievements previously made by Hamadi and Benatallah [49]. We give respective formal definitions of a Web Service and its structural types while striving for unification and recursiveness of given definitions. We deeply investigate formal properties and applicability of various composition patterns and finally develop a Web Service composition algebra to be applied on top of the available composition patterns.

3.1 Web Service Definition

In order to give a definition of a Web service which suits to our methodological purposes and requirements, we have to abstract from the technological understanding of this concept and take a closer look at its functional meaning. A functional definition of a Web service is what we need to adequately model the concept with a formal modeling tool.

Since we intend to use Petri nets as a modeling tool for Web services and Web service compositions, we have to put some requirements onto the Web service definition, which will help us to ensure that services can be correctly modeled with the chosen formalism. The main requirement is that the Web service definition has

to allow representing Web Service as a process of sequential state changes, but not as an inherent structure of the corresponding computational facility. Another important aspect is concurrency. The definition must allow modeling of concurrent processes with the possibility of non-determinism. Finally, the definition has to be recursive in order to treat compositions of services as normal services and as a consequence to allow recursive construction of composite services from other composite services.

So, we define the concept of Web service as follows:

Definition 3.1. *Web Service* is a complex functional unit, which can be conceived as a "black box" having its implicit structure and explicit inputs and outputs, and performing the function F that transforms the input set of arguments X into the output set of outcomes Y.

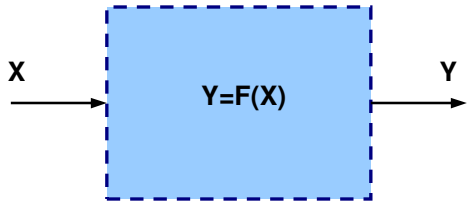

FIGURE 24 Web Service as a "black box"

However, the "black box" representation, shown in Figure 24, does not reveal to us the nature of the service function. From the composition viewpoint it might be important to distinguish between simple and complex service functions in case we want to provide separate definitions for a simple Web Service and a composite Web Service. But we feel that the idea of making a conceptual distinction between simple and composite Web Services is plain, inconvenient and may result in unnecessary ambiguities and other serious problems. Furthermore, such distinction does not fit to our requirement for definition recursiveness.

So, we start from the assumption that all services are potentially composite. Since composite services have to be built recursively from other composite services we need a reference point to start with. As such a reference point we see a primitive generic function that can constitute a whole service alone and can be included into any imaginable service without changing its functionality. Apparently, such function is a completely speculative concept, because a service made of it cannot exist physically. However, it is necessary to introduce this concept for theoretical reasons.

So, we introduce a *zero-function* ϕ and a *zero service* (or empty service) S_0. Zero function is a function that produces no results. Zero service is a service which consists of the only function and this function is zero function, i.e. performs no operation.

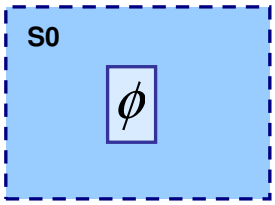

FIGURE 25 Zero function and zero service

However, every formally defined function has to take certain arguments and return certain results. So, the representation of the empty service shown in Figure 25 is not strictly formal. To make it more formal, we say (and this will be justified consequently) that a zero function ϕ takes an argument x and returns x as a result.

$$\Phi(x) = x \tag{3.1}$$

Thus, zero function simply transcribes its input to its output, i.e. does not change the state of the service.

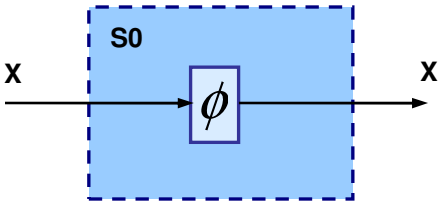

FIGURE 26 Formal representation of zero function and zero service

Analogously, empty service is simply a service that echoes its input parameters to the output, i.e. does not alter them.

From this viewpoint, a normal service shown in Figure 24, which does perform certain operation(s), modifies its input *X* to produce an output *Y*. But this is a simple service: it consists of the only function *F(x)*. Assuming that the function *y=F(x)* is elementary, i.e. performs operation that cannot be further split into sub-operations, we can call this service *atomic*.

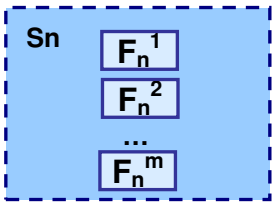

FIGURE 27 Service consisting of a complex function

In principle, any existing service S_n can be represented as containing the only function F_n. But its function would not be elementary for every such service. In the majority of cases service function can be easily decomposed into less complex component functions (see Figure 27).

Now applying the same style of thinking we can say that every component function $F_n^1,\ldots,F_n^m$ can constitute an entire service $S_n^1,\ldots,S_n^m$ respectively. This way we obtain a composite service S_n shown in Figure 27.

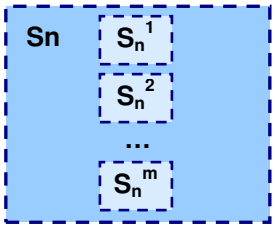

FIGURE 28 Composite service

It is not difficult to assume that the component services $S_n^1,\ldots,S_n^m$ themselves can be composite services if their functions could be similarly split into more primitive ones and if there exist services performing those more simple functions. Such decomposition process can go on until elementary functions are reached and the corresponding atomic services are found. Here the recursiveness of the composite Web service definition becomes evident.

However, before we can speak about a really formal recursive definition, we still need to answer the question of how component services can be composed to constitute another service, i.e. how component services relate to one another. We will get to this point later.

The concepts of the service and the service function, which we introduced, and the way we operate them may raise a logical question: what is the real difference between the service and the service function? Indeed, while defining a composite service we simply substituted component service functions with corresponding component services. But we emphasize again that our constructs are somewhat speculative. So, we intentionally distinguish services and service functions to facilitate the explanation of our cogitative constructions. Secondly, we make such distinction because a function is something implicit and inherent to a service; it cannot be considered and disposed in isolation from the service. Conversely, service is an independent unit that is fully accessible for external operations. Finally, while service function is rather a theoretical concept, service does physically exist. This implies that not every function has a physical counterpart in the form of service. But whenever such counterpart exists the function can be easily substituted with the corresponding service.

So, we conclude that a service consists of service functions and/or component services. These constructs are the elements from which the service structure is built.

But structure is basically defined not only by its elements but also by the relations between its elements. Thus, defining an appropriate set of logical and causal relations between the elements of the service structure will let us conclude the service definition.

Let us for the sake of uniformity refer to service functions and component services as service elements. We specify the following basic relations between service elements:

- *Sequence*. One service element is executed after another. This relation is illustrated in Figure 29.

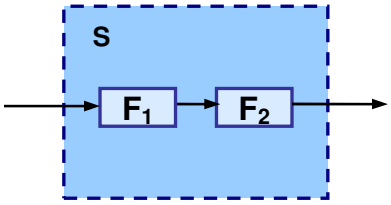

FIGURE 29 Sequence of service elements

- *Mutual exclusion.* Either one service element is executed or another. This relation is illustrated in Figure 30.

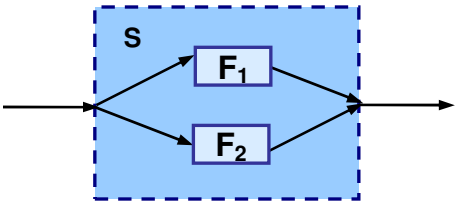

FIGURE 30 Mutual exclusion of service elements

- *Parallelism.* Both service elements are executed in parallel. This relation is illustrated in Figure 31.

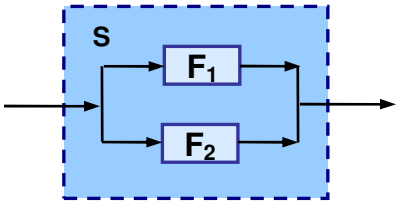

FIGURE 31 Parallelism of service elements

- *Iteration.* One service element is executed multiple times in a row. This relation is illustrated in Figure 32.

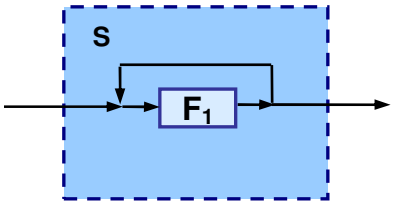

FIGURE 32 Iteration of a service element

By introducing the described relations we assume that a composite service built by composing component services with the help of these relations is also a service. Certainly, we cannot pretend that these four basic relations are adequate for building whatever type of complex relationship that might appear necessary to create a complete service composition logic, but we believe that they are sufficient for formalization in most cases.

For pictorial clarity we also have to add a symbolic notation, describing the syntax of our Web Service algebra and in particular the established relations. We denote the basic relations as follows:

- $\triangleright$ is the *sequence* operator;
- $\otimes$ is the *mutual exclusion* operator;
- $+$ is the *parallelism* operator;
- α is the *iteration* operator.

Let us see how a more complex relation can be formalized on the basis of the given basic relations. For example, we need to build an *unordered sequence* of two services. So, either service S_1 is followed by service S_2 or vice versa. Figure 33 shows the representation of this complex relationship.

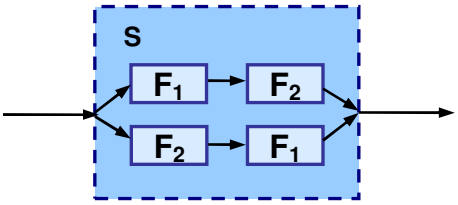

FIGURE 33 Unordered sequence of two service element

We can see that the unordered sequence is built by combining two ordinary sequences under one parallelism as shown by formula (3.2).

$$S : (F_1 \triangleright F_2) \otimes (F_2 \triangleright F_1) \tag{3.2}$$

Thus, we used two basic relations to build a more complex one and thus ensured that the obtained composite service is a service according to our definition.

Now we can at last finalize our recursive definition of Web Service. Therefore, we define Web service as follows:

1. Empty service S_0 is a service.
2. Atomic service S_A is a service.
3. A composite service $S \triangleright S$ is a service.
4. A composite service $S \otimes S$ is a service.
5. A composite service $S + S$ is a service.
6. A composite service αS is a service.
7. Any composite service being the result of applying algebraic operators $\triangleright, \otimes, +, \alpha$ to other services is a service.

As it was already mentioned, we seek for such service definition that would allow representation of a service as a process of sequential state changes. From this viewpoint, semantics of a service operation is absolutely beyond our interest. Neither are we interested in what kind of data a service consumes and produces. The only important thing is the set of service states and the principle of how these states can be reached.

Web Services Glossary [48] denotes a *state* as a set of attributes representing the properties of a component at some point in time. We deepen this definition as follows:

Definition 3.2. *A state of a Web Service is a unique set of attributes' values that describe the process (flow) of Web Service execution at any given point in time.*

However, we require a definition of a Web Service that includes not only states but also these states' changes. To achieve this goal, we also introduce the concept of service operation:

Definition 3.3. *A Web Service operation is an action which switches the service from one state to another. Each service operation has at least one input service state, which is considered a prerequisite for launching the operation, and at least one output service state, which is associated with the effect of the operation execution. A service operation can be performed whenever its input state is active. The operation deactivates its input state and activates its output state on being executed.*

In this context, a Web service can be defined as a partially ordered set of operations. Each operation results in a change of the service state. So, the service can be viewed as a sequence of service states connected to one another through a set of service operations. Basically only one state in such sequence is active at any moment (if the service contains parallel processes there can be more active states, each of which corresponds to a separate process). Every operation has an input state and an output state. An operation can be launched if and only if its input

state is active. When an operation is being executed it deactivates its input state and activates its output state.

Every service has an initial state and a final state. The service can be started whenever its initial state is activated. The service is completed when its final state is reached. The final state is reachable from the initial state of the service through performing one or more operations and possibly proceeding through a number of transient states.

An operation can be understood as a service function, which changes the state of the service. If a certain service function can be mapped to a separate service, this service can be inserted in the place of the corresponding service operation.

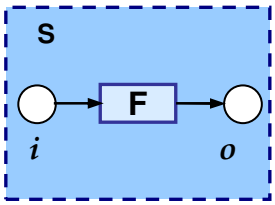

FIGURE 34 Atomic service as a process of state changes

Figure 34 shows an atomic Web service as a sequence of states and operations. When the initial state *i* of the service is active, function *F* can be performed. Accomplishment of *F* transfers the service S into its final state *o*.

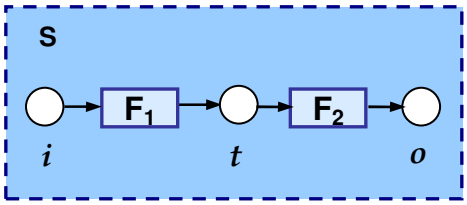

FIGURE 35 A service with two operations and one transient state

Figure 35 shows a composite service performing a sequence of two operations. While in its initial state *i* it accomplishes function F_1 and proceeds to the transient state *t*. In the state *t* function F_2 becomes enabled. After its accomplishment the service reaches its final state *o*.

Since we represent a Web service in a behavioral context and describe its behavior as a partially ordered set of states and operations, an excellent if not the best technique for Web service modeling is Petri nets. In the next section we will show how to model the concept of Web service with the aid of Petri nets formalism.

3.2 Web Services as Petri Nets

Petri nets is a well-founded process modeling technique that has formal semantics [89]. It is particularly suitable for modeling processes that establish parallelism, concurrency and non-determinism. These process characteristics are especially interesting for us, since we assume that composite Web services represent complex processes where component processes can be parallel or concurrent while possible concurrencies and conflicts between them can be resolved in a non-deterministic manner.

An introduction and necessary background information on Petri nets modeling formalism can be found in Chapter 2.

As we stated in the previous section, Web services' execution flows can be seen as a partially ordered sets of service states and service operations. This led us to the idea that Web services should be modeled using two types of elements: states and operations. Furthermore, it has been clearly shown that states are separated from each other and can be connected only via operations. Thus, a Web service structure includes two sets of elements and the elements of one set can be linked only to elements of the other set and vice versa. It can be easily noted that such structural properties are exactly what Petri nets modeling formalism offers.

Being a graph-based visual modeling technique, Petri nets possess two sets of nodes: places and transitions. Places are used to represent states of processes, while transitions model operations. Places cannot be linked directly to other places – they can be only linked to transitions. The same is true for transitions too. So, a Petri net is basically a directed bipartite graph having two types of nodes. Directed arcs in a Petri net graph are used for representing causal relations. This is precisely what we need for appropriate modeling of Web services as processes of state changes.

Thus, a Web service's execution flow (or let us just say Web service for the sake of simplicity) is represented by a Petri net. We will call this Petri net a *service net*. We assume that a service net has one input place that corresponds to the service's initial state and one output place that corresponds to the service's final state. An input place has no incoming arcs, while an output place has no outgoing arcs associated with it. An input place is usually associated with the state of the service at which it is ready to be started. An output place commonly reflects the state at which a service is completed. A service net is organized so that its input and output places are linked to each other through a chain of transient states, possibly multiple. This ensures that the final state of a service is always potentially reachable, i.e. a service can be successfully completed under default conditions.

A service state is active when there is a token in its corresponding place of a service net. Operation of a service begins when there is a token in the input place of a service net, and ends as soon as a token is found in its output place. So, token

is a sort of service state indicator. In the simplest case only one place of a service net can be occupied by a token. A transition, which input place is occupied by a token, is said to be enabled, so it may fire. On firing, a transition removes a token from its input place and deposits it to its output place. This reflects the mechanism of service state changes.

Now we give the definition of a service net. Ordinary Petri nets are usually used for Web service modeling. By ordinary Petri nets we mean Petri nets which do not pose any restrictions on process modeling strength, e.g. number of tokens simultaneously present in a net is not limited, conflicts are allowed, etc. The only restriction is that a place can be linked to a transition by exactly one arc, and vice versa. In other words, the restriction is put on the arcs weights: $\forall f \in F, W(f) = \{0,1\}$. Such Petri nets are also called Place/Transition nets.

Definition 3.4. So, a *service net* is a Petri net defined by a tuple $SN = (P, T, F, i, o)$, where:

- P is a finite set of places;
- T is a finite set of transitions, such that $P \cap T = \varnothing$;
- F is a flow relation; i.e. a set of directed arcs $F \subseteq (P \times T) \cup (T \times P)$;
- i is the input place such that $\forall f \in F, f \notin I(i)$, i.e. $I(i) = \varnothing$ (where $I(i)$ is the input function of the place i);
- o is the output place such that $\forall f \in F, f \notin O(o)$, i.e. $O(o) = \varnothing$ (where $O(o)$ is the output function of the place o);

Now a Petri net-based definition of a Web service can be given according to [49].

Definition 3.5. A *Web service* is a tuple S=(N, D, L, URL, CS, SN) where:

- N is the name of the service, used as its unique identifier;
- D is the service description;
- L is the location of the service;
- URL is the invocation of the service;
- CS is a set of component services constituting the service. If CS={N}then S is an atomic service. Otherwise it is a composite service;
- $SN = (P, T, F, i, o)$ is the service net modeling the service S.

The initial marking M_0 of SN has the only token in the input place i.

In the next section we will describe how to build services using the Petri nets formalism.

3.3 Web Service Algebra

Here we describe the actual process of composition of Web services using Petri nets. Along with the extended syntax of our service algebra, the composition operators described above will be given formal semantics. We will first give detailed descriptions of basic algebraic constructs followed by examples. After that we will propose some advanced composition constructs, which can be, however, in many cases formalized with the help of the basic ones, and investigate their applicability for modeling specific service scenarios. Finally, we will discuss algebraic and non-algebraic properties of the presented composition technique and say some words about service analysis.

3.3.1 Basic Constructs

3.3.1.1 Empty Service

The empty service S_0 is the service carrying zero function, i.e. performing no operation, as we already stated. Although this service cannot physically exist, we use it for theoretical reasons and it is easily modeled by a Petri net.

Definition 3.6. The empty service S_0 is a tuple $S_0 = (N, D, L, URL, CS, SN)$ where:

- N = Empty;
- D = "Empty Web Service";
- L = Null;
- URL= Null;
- CS = {Empty};
- $SN = (\{p\}, \varnothing, \varnothing, p, p)$.

Figure 36 shows the graphical representation of the empty service S_0.

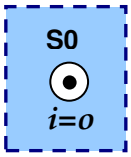

FIGURE 36 Empty service

When we gave the definition of the empty service, we said that this service, firstly, performs no operation, and secondly, does not change the state, i.e., the final state of this service is identical to its initial state. Obviously, performing no operation means that the associated Petri net must not contain transitions. As a matter of fact, the Petri net in Figure 36 contains no transitions. Also, this Petri net contains only

one place being simultaneously the input and the output place of the net. This fact ensures service state preservation property.

3.3.1.2 Atomic Service

An atomic Web service is a service which performs an elementary operation, i.e., an operation that cannot be further decomposed into more primitive operations. Atomic service is a strict concept. Being correctly modeled, its service net must contain exactly one input place, one output place and one transition (which are required by the standard form of Petri net models and ensures that the execution process has a single entry point and a single point of termination). An atomic service performs an operation that changes the initial state of the service directly to its final state.

Definition 3.7. The atomic service S_A is a tuple $S_A = (N, D, L, URL, CS, SN)$ where:

- N is the name of the service;
- D is the service description;
- L is the location of the service;
- URL is the invocation of the service;
- $CS = \{N\}$;
- $SN = (\{i_A, o_A\}, \{t_A\}, \{(i_A, t_A), (t_A, o_A)\}, i_A, o_A)$.

Figure 37 shows the graphical representation of the atomic service S_A.

The service net of the atomic service S_A cannot contain more than one transition because in case of even two transitions potentially it might be possible to find separate atomic services, each of which would perform the corresponding operation in isolation, and build a composite service from those atomic services. However, atomic service's main distinction from other services is its indivisibility.

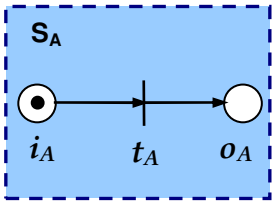

FIGURE 37 Atomic service

Let us show an example of the simplest atomic service and illustrate it with a service net. Imagine that there is a service which obtains two integer values as its input, adds them up and returns their sum. Such a service is modeled by the service net shown in Figure 38.

As we can see from the above example, the service S_{Add} performs an elementary operation of addition. This operation cannot be basically decomposed into a more primitive operation. In Chapter 4 we will give practically justified definitions of decomposability and elementary quality and contemplate a decomposition algorithm dealing with fuzziness of the "elementary quality" concept. The service can be started when input parameters *A* and *B* are received. This corresponds to the initial state *i* of the service. The service is completed when the sum *C* of the parameters *A* and *B* is calculated. This corresponds to the final state *o* of the service.

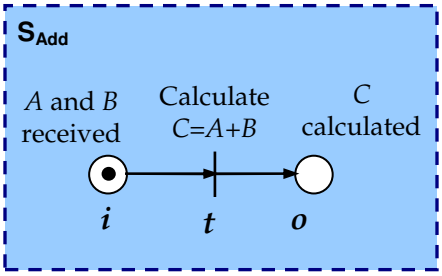

FIGURE 38 Example of atomic service

3.3.1.3 Simple Service

A simple service is a service which does not contain any component service and performs multiple operations or at least one non-elementary operation.

Obviously, a simple service is not a composite one, since it does not include other services in its structure. On the other hand, neither is it atomic, because it contains a number of operations, and every component operation can potentially constitute a single service. So, although simple service is not strictly composite, it can be transformed into a composite service, which is not the case for atomic services. As for a simple service containing only one operation, its main difference from an atomic service is that this operation is non-elementary, i.e., it can be decomposed into two or more component operations.

Definition 3.8. The simple service *S* is a tuple *S* = (*N, D, L, URL, CS, SN*) where:

- *N* is the name of the service;
- *D* is the service description;
- *L* is the location of the service;
- *URL* is the invocation of the service;
- *CS* = {*N*};
- $SN = (P, T, F, i, o)$.

Figure 39 shows the graphical representation of the simple service *S*.

As we can see from Figure 39 a simple service may facilitate the same type of algebraic operators (sequence, mutual exclusion, etc.) that we defined from composite services. However, while in a composite service these operators are applied to component services, in a simple service they are applied to component operations instead. It is then obvious that a simple service can generally be turned into a composite service by substituting its component operations with corresponding component services. We could say that such a substitution trick corresponds to another composition operator called refinement.

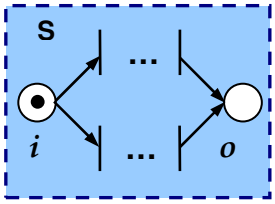

FIGURE 39 Simple service

Let us now illustrate the concept of a simple service with an example. Imagine that there exists a service that calculates distance between two arbitrary cities. How is such service working in general? First, it gets the names of the two cities as its input data. Then it receives their geographical coordinates from a certain database and transforms them into Cartesian coordinates. Having Cartesian coordinates it finally applies the following formula to calculate the distance between cities *A* and *B* with the coordinates (x_A, y_A) and (x_B, y_B) respectively:

$$D = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \tag{3.3}$$

The service net of such distance calculation service is presented in Figure 40.

Although the distance calculation service in Figure 40 may look somewhat sophisticated, it is a simple service according to our classification. All of the component operations performed by the service constitute an independent service functionality and none of them is performed by a component service. Nevertheless, there might exist services that have the same standalone functions as this service's component functions. Inserting those services in the distance calculation service instead of its component operations will create a composite distance calculation service.

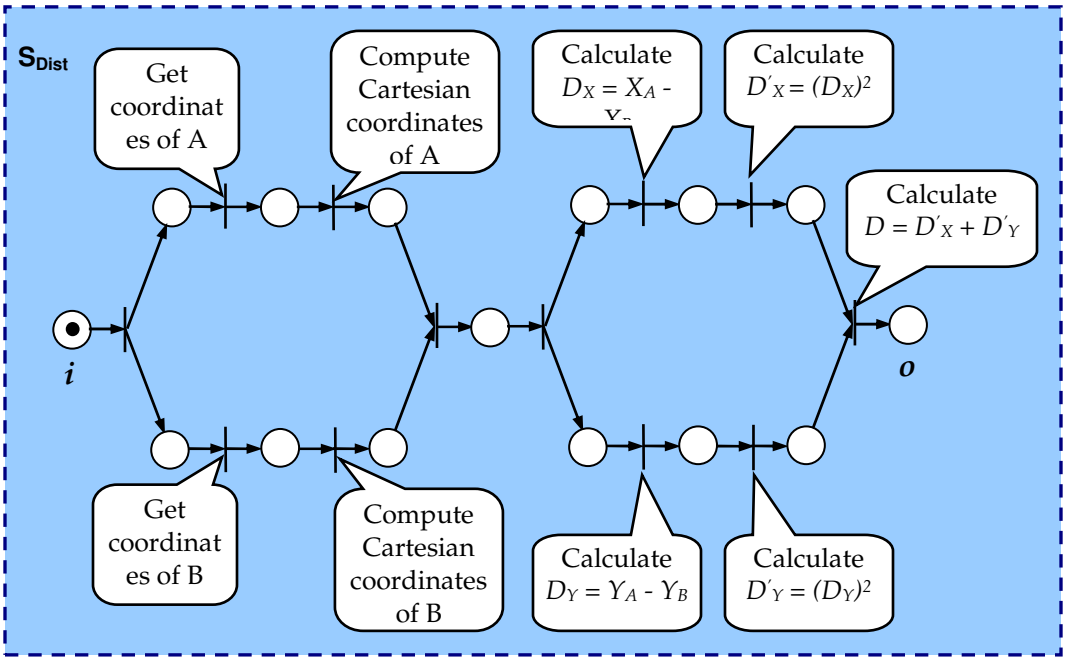

FIGURE 40 Example of a simple service

3.3.1.4 Composite Services

A *composite* service is basically a service which consists of other services. A composite service generally consists of one o more component services, of some auxiliary linkage functionality needed for appropriate composition and not necessarily for its own service functionality.

Composite service as a class of services can be roughly split into two subclasses. One subclass can be called *purely composite services* and the other – *partially composite services*.

Purely composite service is a service which establishes no exclusive functionality, i.e. all the service operations performed by the service are actually performed by its component services. The composite service itself performs only auxiliary operations to link component services together in a correct way. For instance, if we substitute all the labeled operations of the service from Figure 40 with appropriate services, we will obtain a purely composite service. Purely composite services are usually created when building a composite service from scratch, because it is easier to discover an appropriate combination of ready-made services than create that specific service functionality. And of course, this method of building composite services is especially effective in respect to automatic service composition.

Partially composite service is a service which establishes its own exclusive functionality partially refined by other component services. Such composite service does certain part of the job by itself, while the other part for certain reasons is made up by its component services. For example, the service in Figure 40 can be easily made partially composite if we substitute a couple of its component operations with suitable services. This is usually a rather effective method for already existing services. Say, however, that we are not satisfied with the accuracy of Cartesian coordinates calculation provided by our distance calculation service (see Figure 40), and we have a standalone service transforming geographical coordinates into Cartesian coordinates with much better accuracy. What should we do in that case? We take this specific transformation service and insert it inside our distance calculation service instead of its corresponding operation. So, we obtain a new composite service making essentially the same things but with better quality.

Below we give formal definitions for service composition operators and constructs, which can be considered the basic building blocks of composite services. Almost all of these constructs are used to build purely composite services, and only the refinement operator helps create partially composite services.

3.3.1.5 Sequence

The operator of sequence helps building a composite service out of two (or more) component services. The component services linked with the sequence operator are executed in sequence, i.e. one after another. The sequence operator is not commutative, i.e., the order of the sequence is strict:

$$S_1 \triangleright S_2 \neq S_2 \triangleright S_1 \tag{3.4}$$

Definition 3.9. The composite service *S* being the sequence of two component services $S_1 \triangleright S_2$ is a tuple *S* = (*N, D, L, URL, CS, SN*) where:

- *N* is the name of the composite service;
- *D* is the composite service description;
- *L* is the location of the composite service;
- *URL* is the invocation of the composite service;
- $CS = CS_1 \cup CS_2$;
- $SN = (P, T, F, i, o)$ where:
  - $P = P_1 \cup P_2$;
  - $T = T_1 \cup T_2$;
  - $F = F_1 \cup F_2 \cup \{(o_1, t), (t, i_2)\}$;
  - $i = i_1$;
  - $o = o_2$.

Figure 41 shows the graphical representation of the composite service $S = S_1 \triangleright S_2$.

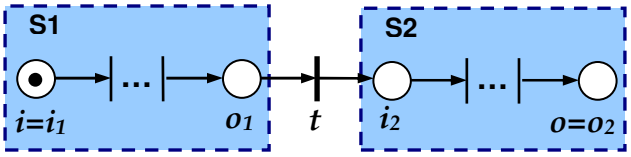

FIGURE 41 Sequence of two services.

A typical example of a composite service built in a sequential manner is a supply chain. For instance, we can imagine a ticket booking service or some software download service, where the customer first books tickets or requests software, then pays, and after that the service provider delivers tickets or software to him. Each of these operations can be, and sometimes should be, performed by a separate service. It is quite common that a provider of a certain type of services has neither qualification, nor capacity, nor sometimes even rights to provide other types of services, which however are vitally important for appropriate provision of the main service value. So, such provider may simply outsource this auxiliary but still necessary functionality to other service providers. For example, a travel agency does not have rights to directly sell airplane tickets, but it can easily use the appropriate services of various airlines and incorporate them within their traveling services. Neither can that agency provide electronic payment facilities. This is the prerogative of banks. Again, bank payment services can be incorporated to some traveling services to facilitate better service usability and smoother user experience

Sequence operator is usually the easiest and perhaps the most common way of composing such diverse but supplemental services in a single supply chain.

From a different perspective the sequence operator is always used when services are causally sequential, i.e. one service cannot start until the other is over. Such dependence is usually justified by certain logical criteria, such as nothing can be provided until it is paid. This type of causality may be called *logical* causality. Technically, this can be reflected by appropriate preconditions established by services and post-effects generated by them. For example, a payment shouldn't be made until tickets are booked, or download must not be instantiated until money is transferred to the provider's account. Another type of causality is when a subsequent service uses in its operation data produced by the preceding service. This case might be called *functional causality*. This type of relationship is technically reflected in services by required data inputs and produced data outputs. Within the distance calculation service (see Figure 40) the actual computation of the distance from Cartesian coordinates cannot start until the geographical coordinates are transformed to Cartesian ones.

Figure 42 shows a composite ticket purchase service built as a sequence of three component services. Three component services constitute the basic supply

chain. However, all the three services are owned by different service providers. Still this fact does not prevent a new composite value-added service from being composed. Double application of the sequence operator here is justified by logical and, perhaps, functional causality which exist between the component services. Payment process starts as an obvious consequence of the completion of the ticket booking procedure. Again, the delivery service naturally has the condition of payment completion among its prerequisites.

This example also illustrates some of the properties of the sequence operator. As we already mentioned, it is not commutative, that is, the order of service execution is important, which is explained by the causality relations between the services. The sequence operator possesses, on the other hand, the associativity property:

$$(S_1 \triangleright S_2) \triangleright S_3 = S_1 \triangleright (S_2 \triangleright S_3) \tag{3.5}$$

This property is useful since it allows an easy extension to the number of operands linked by sequence operators. Thus, we can easily build a sequence of multiple services using a binary operator.

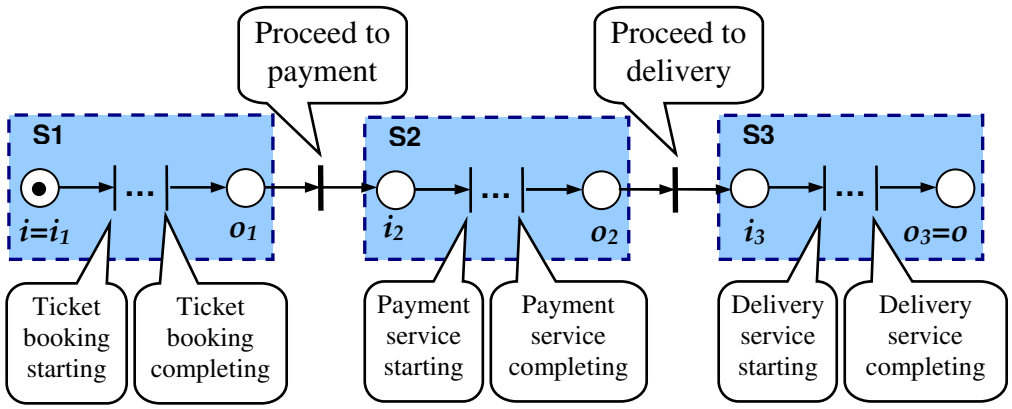

FIGURE 42 Tickets purchase composite service

3.3.1.6 Mutual Exclusion

The mutual exclusion (also often called *alternative*) operator allows building a composite service out of two (or more) services. The component services linked by the mutual exclusion operator are executed alternatively, i.e., either one service is executed or another, but not both.

Definition 3.10. The composite service S being the mutual exclusion of two component services $S_1 \otimes S_2$ is a tuple $S = (N, D, L, URL, CS, SN)$ where:

- N is the name of the composite service;
- D is the composite service description;
- L is the location of the composite service;
- URL is the invocation of the composite service;
- $CS = CS_1 \cup CS_2$;
- $SN = (P, T, F, i, o)$ where:
  - $P = P_1 \cup P_2 \cup \{i, o\}$;
  - $T = T_1 \cup T_2 \cup \{t_{i1}, t_{i2}, t_{o1}, t_{o2}\}$;
  - $F = F_1 \cup F_2 \cup \{(i, t_{i1}), (i, t_{i2}), (t_{i1}, i_1), (t_{i2}, i_2), (o_1, t_{o1}), (o_2, t_{o2}), (t_{o1}, o), (t_{o2}, o)\}$.

Figure 43 shows the graphical representation of the composite service $S = S_1 \otimes S_2$.

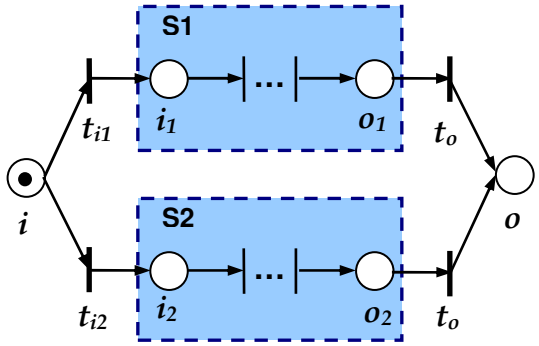

FIGURE 43 Mutual exclusion of two services

The mutual exclusion composition construct has basically two main use cases. One is presenting alternate functionality. This means that a composite service has a goal to be achieved as a result of its execution, but it may appear that the goal is achievable, say, in two different ways implying completely different sequence of actions and utilizing completely different functionality. In such cases a logical solution is provide, as alternatives, two component services corresponding to the two distinct ways of the goal achievement. This is exactly what the mutual exclusion operator does.

The other use case is about fault tolerance. Services sometimes can be quite unreliable. What to do if a necessary service suddenly and opportunely fails, but we badly need its functionality to complete our service? In such situations we may encounter serious problems finalizing our service. Establishing some redundancy for presumably unreliable component services may prevent the composite service from unanticipated failure due to spontaneous failure of its component service. The mutual exclusion operator can be utilized to model alternative execution of two or more different component services performing the same or very similar functionality, thus reducing the composite service failure probability.

We can see from the formal definition of the mutual exclusion construct that the choice between the alternatives is not determined by any external factors. This plays a very important role in expressing non-determinism trough our Petri net service algebra.

Figure 44 illustrates the example of the service of the first type, i.e. alternative functionality. It is almost the same ticket purchase service as shown in Figure 42. However, it differs from the latter in providing two possible ways of payments. A customer can now pay for booked tickets either using bank transfer or using Pay Pal service. Each of these payment methods is provided by a separate component service. Component payment services are incorporated into the composite service by means of the mutual operator, thus ensuring that a customer pays only once using only one payment method and performing only one transaction. If there are services that allow other payment methods, e.g. credit card, web money, etc., they can be easily incorporated into the composite service using the same mutual exclusion operator.

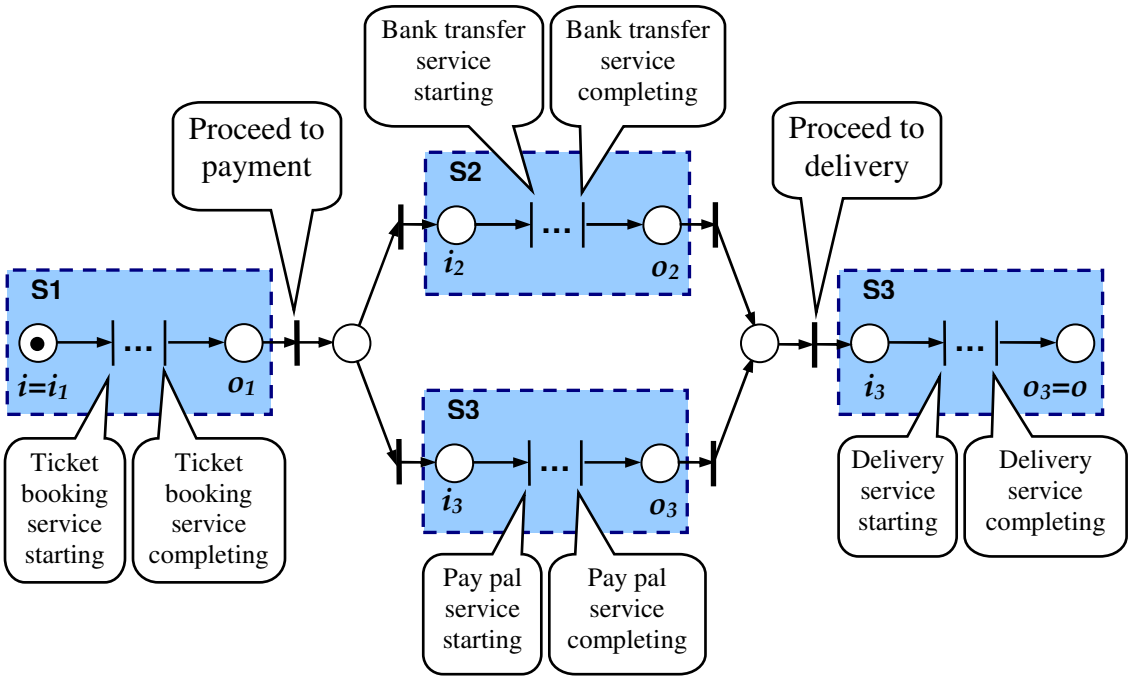

FIGURE 44 Ticket purchase service with alternative payment methods

Certainly, in practice the choice among services that provide alternative functionalities should hardly be random. Such a choice is naturally justified by certain factors. For example, in case of different payment methods the customer's preference is the factor that decides which one of the available methods to choose. Any payment method will hardly fit customer's needs and requirements. For instance, if the customer does not own a credit card he cannot simply be

authorized to use a number of payment methods. So, the selection between alternatives must be made consciously. In some other cases the choice can be made automatically on the basis of certain situational data or the properties of the available services, but still this choice remains deterministic by nature. In order to formalize the deterministic type of choice among different alternatives the mutual exclusion operator has to be either reformulated or supplied with additional control facilities. We will comprehensively discuss the issue of deterministic choice and control over it further in this thesis, but for the moment it is beyond the scope of the mutual exclusion operator and the Web service algebra in their current formalization.

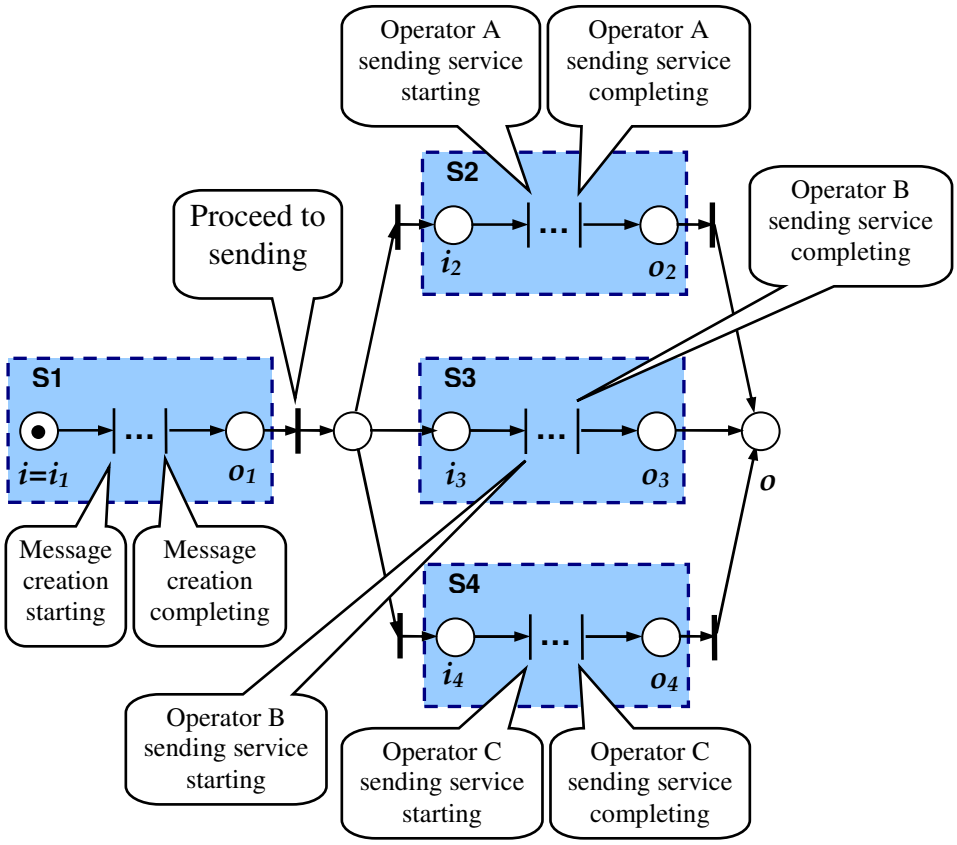

FIGURE 45 Message exchange service with alternative operators

Figure 45 shows another example of the mutual exclusion operator usage. Here we can see another use case related to fault tolerance rather than to alternative functionality. Imagine that a mobile user travels abroad with his mobile phone. Whenever he wants to send a short message from his mobile device, a certain sequence of actions is taken. This set of operations can be described with a service net pictured in Figure 45. Being a foreign user to mobile operators working in the locality, the user consumes the operator services under some roaming agreements.

Several mobile operators can usually provide the user with the access to their services under separate roaming agreements. This feature can be utilized for composing a more reliable message service for the roaming user. All the mobile operators provide more or less the same type of message exchange service, i.e., the functionality varies negligibly among the services provided by different operators. So, combining the message exchange services of different operators under the operator of mutual exclusion, we guarantee a more reliable message exchange service for the roaming user, since the probability that at certain location all the available operators are out of range is substantially lower than for the case of a single operator.

In the case of combining service with the same functionality with the aid of mutual exclusion operator the choice can be often non-deterministic (in the sense that the choice of a particular service is not predefined), because we may see no functional difference among the alternative services and the choice among them can be made based on non-functional characteristics at run-time or even randomly. So, for this type of alternatives the mutual exclusion operator fits ideally.

Speaking about formal properties of the mutual exclusion operator, it should be noted that this type of relation is commutative:

$$S_1 \otimes S_2 = S_2 \otimes S_1 \tag{3.6}$$

and associative:

$$(S_1 \otimes S_2) \otimes S_3 = S_1 \otimes (S_2 \otimes S_3) \tag{3.7}$$

3.3.1.7 Parallelism

The parallelism operator allows building a composite service out of two (or more) services. The component services linked by the parallelism operator are executed concurrently, i.e., both simultaneously and in parallel.

Definition 3.11. The composite service *S* being the parallelism of two component services $S_1 + S_2$ is a tuple *S* = (*N, D, L, URL, CS, SN*) where:

- *N* is the name of the composite service;
- *D* is the composite service description;
- *L* is the location of the composite service;
- *URL* is the invocation of the composite service;
- $CS = CS_1 \cup CS_2$;
- $SN = (P, T, F, i, o)$ where:
  - $P = P_1 \cup P_2 \cup \{i, o\}$;
  - $T = T_1 \cup T_2 \cup \{t_i, t_o\}$;
  - $F = F_1 \cup F_2 \cup \{(i, t_i), (t_i, i_1), (t_i, i_2), (o_1, t_o), (o_2, t_o), (t_o, o)\}$.

Figure 46 shows the graphical representation of the composite service $S = S_1 + S_2$.

The parallelism operator is useful for combining services in two situations. Even if these two use cases are totally dissimilar or even contrary, it is quite logical to apply the parallelism operator in these cases.

One case refers to independent services. Indeed, two services can be executed concurrently if they do not depend on each other, i.e. they are not subjected to causal relationship. Although independent services are surprisingly often combined by the sequence operator (especially designed for formalization of causal dependence), it is quite obvious that concurrent execution of services is in many cases preferable to sequential execution mainly due to reduced execution time. This is the formal part. It is logical that two independent functionally different processes should be combined by the parallelism operator to increase performance. However, functionally similar independent services can also be subjected to concurrent execution, for example, in order to test which of them exhibits better performance characteristics.

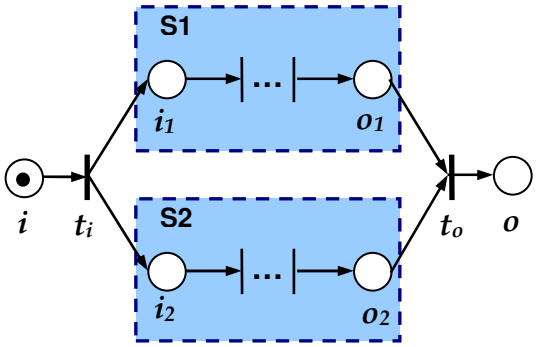

FIGURE 46 Parallelism of two services

Figure 47 shows how we transformed our ticket purchase service (see Figures 42 and 44) into a travel purchase service using the parallelism operator. The most common type of travel consists of two distinct parts: the travel itself and the stay at a new place. Therefore, a travel arrangement consists usually of tickets purchase and accommodation purchase. A good travel purchase service allows making both tickets booking and hotel reservation and then paying once for everything. The processes of ordering tickets and reserving rooms at a hotel are basically independent and can be performed concurrently. That is why the corresponding services are linked with the parallelism operator. However, since the composite travel purchase service implies paying for everything at once, both the hotel booking service and the ticket booking service have to be completed before the payment service can start. And this aspect is also correctly modeled by the parallelism operator.

Another, perhaps, controversial use case of the parallelism operator is when component services are not independent. Moreover, the parallel execution is absolutely necessary whenever two services are mutually dependent. What does mutual dependency mean? One service during its operation may require some data that can be produced by another service and vice versa. So, we have a situation with two correlated services which cannot complete without intermediate data exchange between each other. This case is referred to as *co-execution*. The parallelism operator suits well for such a case. However, it requires certain modifications to facilitate intermediate data exchange between services. That is why in one of the following subsections we will present a special extension of the parallelism operator: parallelism with communication.

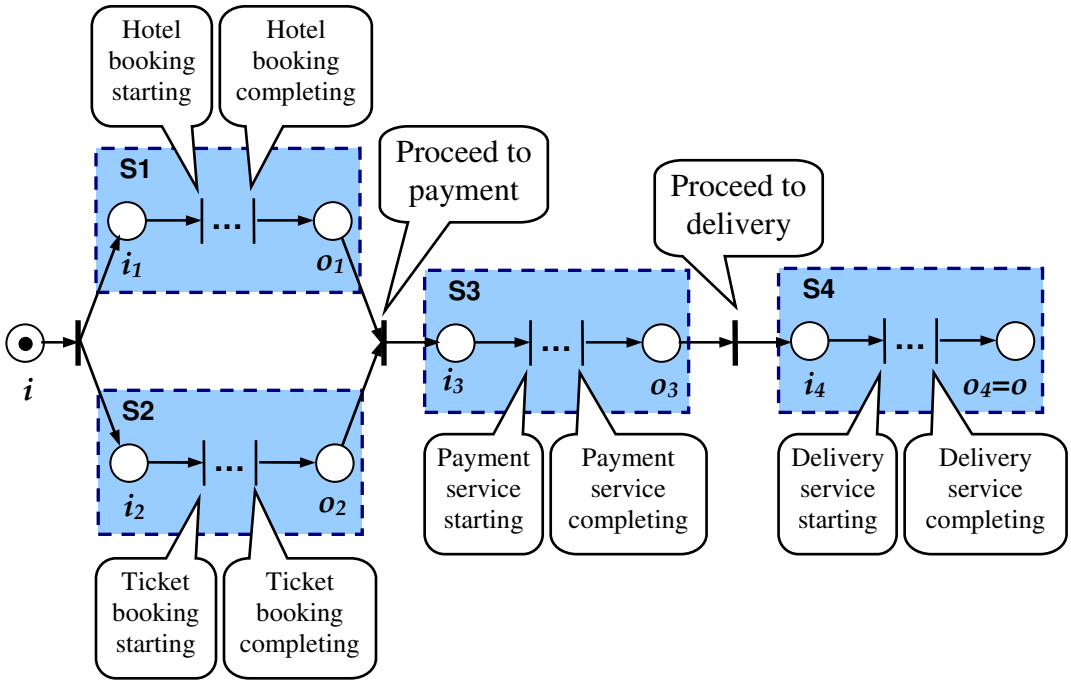

FIGURE 47 Travel purchase service with parallel booking services

As regards formal properties, the parallelism operator holds the properties of commutativity:

$$S_1 + S_2 = S_2 + S_1 \tag{3.8}$$

and associativity:

$$(S_1 + S_2) + S_3 = S_1 + (S_2 + S_3) \tag{3.9}$$

3.3.1.8 Iteration

The iteration (or cycle) operator allows building a composite service from one component service. The execution of the component service subjected to the

iteration operator represents a cyclic iteration of the same service for certain number of times.

Definition 3.12. The composite service *S* being the iteration $S = \alpha S_1$ is a tuple *S* = (*N*, *D*, *L*, *URL*, *CS*, *SN*) where:

- *N* is the name of the composite service;
- *D* is the composite service description;
- *L* is the location of the composite service;
- *URL* is the invocation of the composite service;
- $CS = CS_1$;
- $SN = (P, T, F, i, o)$ where:
  - $P = P_1 \cup \{i, o, c\}$;
  - $T = T_1 \cup \{t_i, t_o, t\}$;
  - $F = F_1 \cup \{(i, t_i), (t_i, i_1), (o_1, t_o), (t_o, o), (o_1, t), (c, t), (t, i_1), (\overline{c, t_o})\}$.

The negation mark over the arc $(\overline{c, t_o})$ in the previous expression identifies an inhibitor arc. Figure 48 shows the graphical representation of the composite service $S = \alpha S_1$.

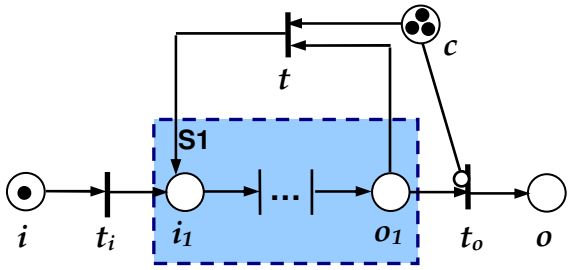

FIGURE 48 Iteration of service

The iteration operator is interesting from the application perspective as well as from the formalization viewpoint. It could be the most original, non-ordinary and ambiguous operator within our service algebra.

Its first peculiarity is that it is a unary operator, while the rest of the operators used are binary. So, it performs a unary operation, an operation which uses a single operand. The iteration operator manipulates a single service, performing it multiple times in a row.

It seems logical that the iteration operator can be decomposed into a linear combination of multiple sequence operators applied in turns to the same service. However, it would allow for correct iteration formalization only in case of infinite number of cycles. But our iteration operator is especially introduced to formalize

the case when the number of iterations is finite. In order to spread linear combination of sequence operators into such case we would need to establish some sort of counter facility, which would determine how many sequence operators should be applied. It seems to be a rather cumbersome way to build a basic type of formalization. Instead, the iteration operator allows for a rather simple formalization of a repeated process while also providing modeling simplicity and pictorial clarity as can be seen in Figure 48.

From the application viewpoint, it is possible to mark out two distinct use cases. The first one is a conveyor or a production line. Such a service repeatedly produces the same type of outcome performing the same set of operations over and over again. This type of application is trivial. Along with the applications of all previously described composition constructs it strives for producing some more complex added value.

Another use case of the iteration operator is more interesting. It can be called *state maintenance*. The meaning of that is that a service is performed iteratively and as a result of its operation it keeps up a certain level of some external factor. For example, imagine a temperature control service being a part of some climate control system. Such a service may contain two opposite services: a heating service and a cooling service. These services are composed into the temperature control service with iteration operators. Whenever the measured temperature deviates from the desired level, the iteration of the corresponding temperature restoration service is started.

The main distinction between the two use cases is the following. In case of conveyor-like functionality a service operates repeatedly but conceptually finite number of times to produce a certain number of identical effects. State maintenance case generally implies that the target state can be maintained infinitely long until some external control influence will interrupt the execution of this iterative process. Obviously, two services different in such a way may also require different modeling. The model shown in Figure 48 suits better for the case of conveyor, while for state maintenance the model presented in Figure 50, which we consider below, is preferable.

Let us graphically illustrate this example. Figure 49 shows the service net of the temperature control service we just described. This is apparently a composite services built out of three component services: temperature measurement service S_1, heating service S_2, and cooling service S_3. Service S_1 measures current temperature within physical neighborhood and calculates the deviation between the current temperature level and the desired temperature level. Whenever the deviation is different from zero it should be compensated. The heating service is used to compensate negative deviations, and the cooling service positive ones.

Both the heating and the cooling service are inserted into the composite service under iteration operators. If a non-zero deviation is observed, it might be sufficient to execute the corresponding compensation service just once to keep up

the level. For correct service operation in other cases iteration operators used. The iteration operator has a special place in its service net, which is allowed to store multiple tokens. This place is labeled as p in the service net shown in Figure 49. The number of tokens in the place p is essentially equivalent to the number of remaining service iterations. However, in order to put those tokens into the place p it is necessary to establish a certain external control mechanism capable of putting a certain number of tokens into the place p with respect to some external factor. In our example such a factor is the temperature deviation: the bigger it is, the more tokens have to be put to the place p, because logically the service should make more iterations to compensate for a bigger deviation.

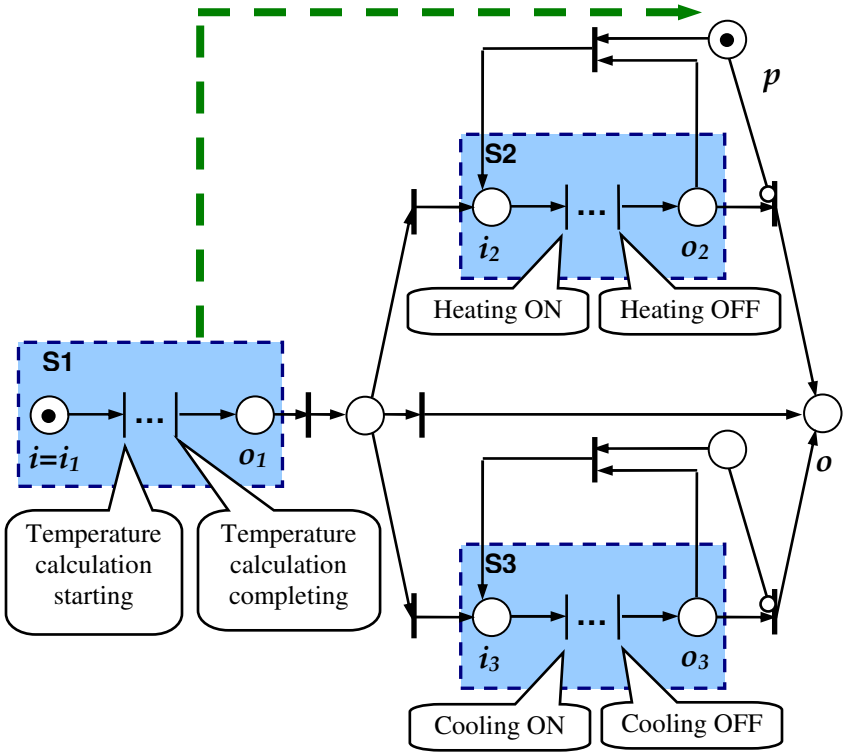

FIGURE 49 Temperature control service

We, however, are interested only in how the service does what it does. So, the external control mechanism is currently beyond our focus. Still, we realize its necessity just as we do in the case of the mutual exclusion operator, where external control is desirable to perform deterministic choices. As a matter of fact, we show a control factor (thick dashed arrow in Figure 49) only because without introducing it the understanding of the iteration operator would be incomplete and, perhaps, incorrect. The place p of the iteration construct with all its incident arcs is more a semantic control structure than an element of building logical relations between services, but we believe that its presence in our formalization is absolutely

necessary in order to present a limited iteration operator, which is much more useful than its unconstrained variant.

Figure 50 shows the unconstrained version of the iteration construct. Obviously, the service net is built to allow non-determinism. Since the place o_1 has two outgoing arcs, we have a conflict, i.e. either the component service will be executed once again or the composite service will be completed. In order to present a deterministic iteration construct we again need some control over the mentioned conflict. Surely, such control can be externally established. But the iteration construct illustrated in Figure 48 is advantageous here, because it not only provides such a control explicitly, but moreover explicitly determines the number of iterations. There are also some other advantages of such formalization, which will be discussed in the following chapters.

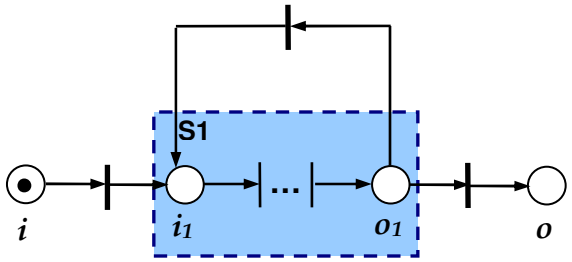

FIGURE 50 Unconstrained iteration of service

We will also further and more thoroughly discuss the issues of determinism and semantic control in the next chapters.

Finally, one more peculiarity of our iteration operator is that along with multiple tokens in one place its service net contains a special advanced Petri net element – an *inhibitor arc* [89] (drawn as an arc ending not in an arrow but in a circle in Figure 48). As we already explained in the previous chapter, inhibitor arcs are the extension of the ordinary Petri nets that increase their modeling power. Inhibitor arcs allow formalization of zero conditions. In our case the inhibitor arc is the only way to facilitate the composite service completion when all tokens in the place *p* are consumed.

3.3.2 Advanced Constructs

Advanced service composition constructs are basically composition constructs that are more sophisticated than the basic ones. Their primary distinction from the basic constructs is that they should not be necessarily used for building composite services. While basic constructs represent the set of necessary algebraic operations over services, advanced constructs just increase the modeling power of our service algebra, which can manage without them in the majority of cases.

Advanced composition constructs can be roughly classified into three main types:

1. Combination of basic constructs with reduction of composition complexity;
2. Extension of basic constructs with additional features;
3. Completely new constructs for modeling of special cases.

It should be noted that the set of the advanced constructs is not fixed. New constructs can be, potentially, designed and added to the service algebra if they appear to conform to our service composition requirements.

Below we present some of the most commonly used or the most interesting advanced composition constructs and operators accompanied with formal definitions.

3.3.2.1 Unordered Sequence

The unordered sequence operator allows building a composite service out of two component services. The services are executed sequentially, one after another, but the order of their execution is not determined.

Definition 3.13. The composite service *S* being the unordered sequence of two component services $S_1 \Leftrightarrow S_2$ is a tuple *S* = (*N, D, L, URL, CS, SN*) where:

- *N* is the name of the composite service;
- *D* is the composite service description;
- *L* is the location of the composite service;
- *URL* is the invocation of the composite service;
- $CS = CS_1 \cup CS_2$;
- $SN = (P, T, F, i, o)$ where:
  - $P = P_1 \cup P_2 \cup \{p_1, p_2, p_3, p_4, p_5\}$;
  - $T = T_1 \cup T_2 \cup \{t_i, t_{i1}, t_{i2}, t_{o1}, t_{o2}, t_o\}$;
  - $F = F_1 \cup F_2 \cup \{(i, t_i), (t_i, p_1), (t_i, p_2), (t_i, p_3), (p_1, t_{i1}), (p_2, t_{i2}), (p_{3,} t_{i1}),$ $(p_3, t_{i2}), (p_3, t_o), (t_{i1}, i_1), (t_{i2}, i_2), (o_1, t_{o1}), (o_2, t_{o2}), (t_{o1}, p_3), (t_{o1,} p_4), (t_{o2}, p_3),$ $(t_{o2}, p_5), (p_4, t_o), (p_5, t_o), (t_o, o)$.

Figure 51 shows the graphical representation of the composite service $S = S_1 \Leftrightarrow S_2$.

The unordered sequence construct is an advanced construct of the first type. So, it is a combination of basic constructs with reduction in the number of component services. More specifically, an unordered sequence $S = S_1 \Leftrightarrow S_2$ is the mutual exclusion of two sequences:

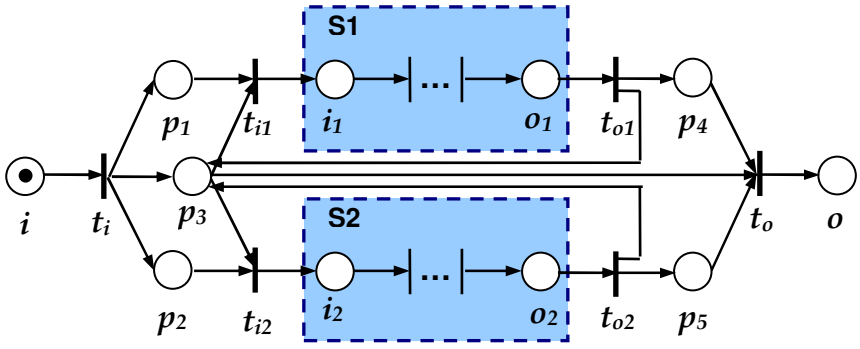

FIGURE 51 Unordered sequence of two services

$$S = S_1 \Leftrightarrow S_2 = (S_1 \triangleright S_2) \otimes (S_2 \triangleright S_1) \tag{3.10}$$

However, unordered sequence composition of two services using only basic operators of sequence and mutual exclusion results in double occurrence of each service within the service net as each service has exactly two entries on the right hand side of the formula (3.10).

Unordered sequence of two services built using basic operators of sequence and mutual exclusion is illustrated in Figure 52.

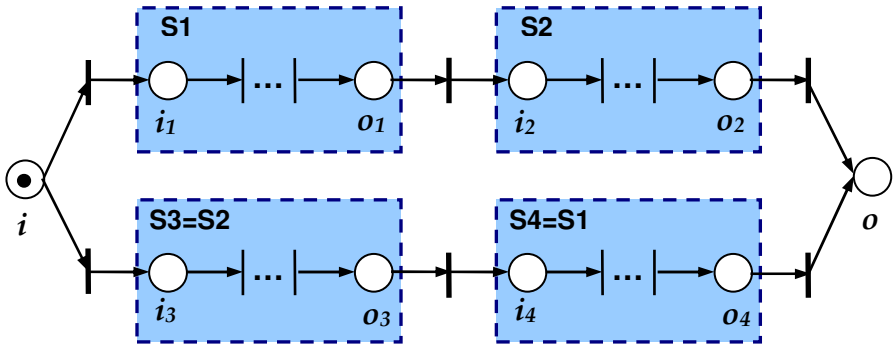

FIGURE 52 Unordered sequence of two services using basic operators

It can be easily seen from the service net presented in Figure 52, that such a way of building unordered sequence composition demonstrates redundancy. Both of the component services are included in the service net twice. Even though such redundancy is quite unimportant from the service operation viewpoint, from the viewpoint of service composition modeling it is simply inefficient. At least it looks cumbersome in comparison to the service net model shown in Figure 51. The introduction of the separate unordered sequence operator, which does effectively the same as the alternative of two sequences does, but reduces the number of

component services' entries in a composite service net, is justified by its elimination of structural redundancy. Component services linked in an unordered sequence cannot functionally depend on one another. If that were the case, arbitrary order of execution of two services would not be acceptable. So, two services can be either independent of each other, or logically correlated.

In case of total independence of two services the unordered sequence operator becomes an alternative to the parallelism operator. For example, there may be a requirement that services should not be executed concurrently, which means that the parallelism operator cannot be used. In such a situation it can be easily substituted by the unordered sequence operator. The parallelism is superior to the unordered sequence from the performance viewpoint, since for parallel execution the execution time is significantly less than for sequential execution. However, the advantage of sequential execution is lesser resource consumption. So, the use of the unordered sequence operator instead of the parallelism operator can be justified in cases where execution time is not as important as avoidance of resource saturation. For example, if there are two independent services which should both be executed on the same computer and are so resource demanding that each consumes the whole memory and processor throughput, it is, obviously, beneficial to execute services in sequence rather than in parallel.

Another interesting case of the unordered sequence operator application is when services are logically and mutually dependent, while still being functionally independent, i.e. two services *complement* each other. For instance, a customer might need to purchase two types of goods, which are different but useless without each other. Then parallel orders for purchasing those goods is a more risky way of making the purchase, because it might appear during the execution that one of the goods is unattainable for certain reasons. As a result, the whole service will fail and the customer will pay for both of its component services. Nevertheless, if the component purchase services are linked by the unordered sequence operator, the anticipated cost of the service failure is fifty percent less, because the probability that in case of service failure the first service in sequence fails is 1/2.

So, let us illustrate the use of the unordered sequence operator by the example of our travel purchase service (see Figure 47). Within that composite service there are two component services, namely, the ticket booking service and the hotel booking service, which complement each other. Apparently, hotel reservation is useless without the possibility to reach it, and vice versa. Figure 53 illustrates the corresponding service net of the modified travel purchase service.

The formal properties of the unordered sequence operator are also rather interesting. It is commutative since $S_1 \Leftrightarrow S_2 = S_2 \Leftrightarrow S_1$, but not associative because $(S_1 \Leftrightarrow S_2) \Leftrightarrow S_3 \neq S_1 \Leftrightarrow (S_2 \Leftrightarrow S_3)$. At first glance it might seem that the unordered sequence operator should hold the associativity property, because its ordered variant, the sequence operator, does hold this property. However, as we can see

from Figure 53, in contrast to the sequence operator the unordered sequence construct does not link services in a linear fashion. Now let us verify these properties in a formal way.

We claim that the unordered sequence operator $S_1 \Leftrightarrow S_2$ is commutative because the following algebraic transformations are valid:

$$S_1 \Leftrightarrow S_2 = (S_1 \triangleright S_2) \otimes (S_2 \triangleright S_1), \tag{3.11}$$

$$S_2 \Leftrightarrow S_1 = (S_2 \triangleright S_1) \otimes (S_1 \triangleright S_2). \tag{3.12}$$

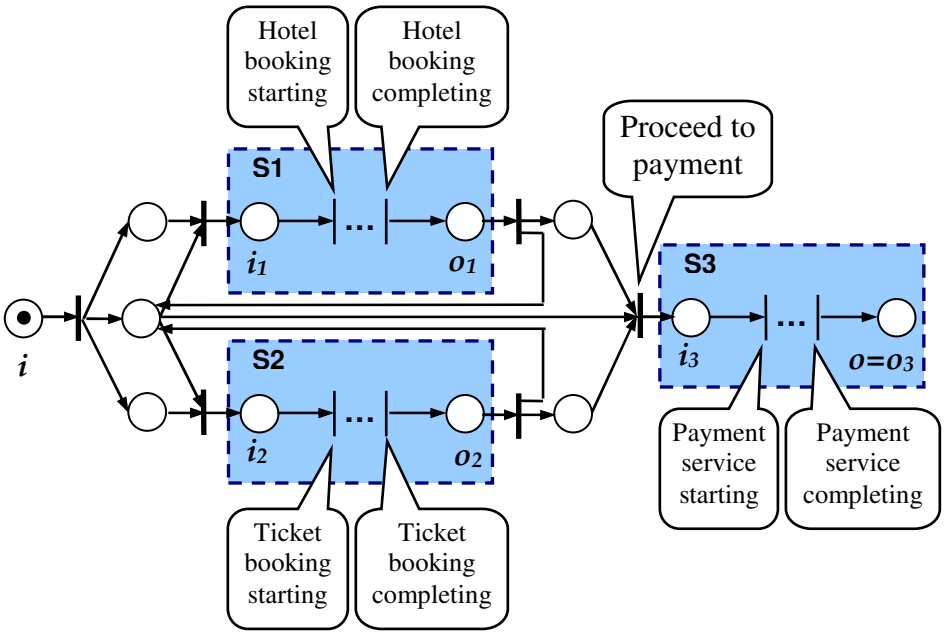

FIGURE 53 Optimized travel purchase service

Since the mutual exclusion operator is commutative by the definition; we have:

$$(S_1 \triangleright S_2) \otimes (S_2 \triangleright S_1) = (S_2 \triangleright S_1) \otimes (S_1 \triangleright S_2), \tag{3.13}$$

which is equivalent to:

$$S_1 \Leftrightarrow S_2 = S_2 \Leftrightarrow S_1. \tag{3.14}$$

Now we prove that the unordered sequence operator does not hold the associativity property (3.15). Let us use the rule of contraries. Assume that the property (3.15) holds.

$$(S_1 \Leftrightarrow S_2) \Leftrightarrow S_3 = S_1 \Leftrightarrow (S_2 \Leftrightarrow S_3) \tag{3.15}$$

First, we take the left-hand side of the equation (3.15) substitute the unordered sequence operators according to (3.11).

$$(S_1 \Leftrightarrow S_2) \Leftrightarrow S_3 = ((S_1 \triangleright S_2) \otimes (S_2 \triangleright S_1)) \Leftrightarrow S_3 \quad (3.16)$$

$$(S_1 \Leftrightarrow S_2) \Leftrightarrow S_3 = (((S_1 \triangleright S_2) \otimes (S_2 \triangleright S_1)) \triangleright S_3) \otimes (S_3 \triangleright ((S_1 \triangleright S_2) \otimes (S_2 \triangleright S_1))) \quad (3.17)$$

Then we open parentheses on the right-hand side of the equation (3.17) and according to the distribution rule we obtain:

$$(S_1 \Leftrightarrow S_2) \Leftrightarrow S_3 = \underbrace{(S_1 \triangleright S_2 \triangleright S_3)}_{L_1} \otimes \underbrace{(S_2 \triangleright S_1 \triangleright S_3)}_{L_2} \otimes \underbrace{(S_3 \triangleright S_1 \triangleright S_2)}_{L_3} \otimes \underbrace{(S_3 \triangleright S_2 \triangleright S_1)}_{L_4} \quad (3.18)$$

Thus, equation (3.18) shows that the left-hand side of the formula (3.15) $(S_1 \Leftrightarrow S_2) \Leftrightarrow S_3$ equals to one of the four possible sequences of services S_1, S_2 and S_3 labeled as L_1, L_2, L_3 and L_4 respectively.

Now, applying the same type of algebraic transformation to the right-hand side of the equation (3.15) we obtain:

$$S_1 \Leftrightarrow (S_2 \Leftrightarrow S_3) = \underbrace{(S_1 \triangleright S_2 \triangleright S_3)}_{R_1} \otimes \underbrace{(S_1 \triangleright S_3 \triangleright S_2)}_{R_2} \otimes \underbrace{(S_2 \triangleright S_3 \triangleright S_1)}_{R_3} \otimes \underbrace{(S_3 \triangleright S_2 \triangleright S_1)}_{R_4} \quad (3.19)$$

Formula (3.19) shows that the right-hand side of the formula (3.15) $S_1 \Leftrightarrow (S_2 \Leftrightarrow S_3)$ also equals to one of the four possible sequences of services S_1, S_2 and S_3 labeled as R_1, R_2, R_3 and R_4 respectively.

If we now compare the right-hand sides of formulas (3.18) and (3.19) we easily see that while two of their sequences are equivalent between the formulas, the other two differ from one another.

$$L_1 = R_1 = S_1 \triangleright S_2 \triangleright S_3 \quad (3.20)$$

$$L_4 = R_4 = S_3 \triangleright S_2 \triangleright S_1 \quad (3.21)$$

$$L_2 = S_2 \triangleright S_1 \triangleright S_3 \neq S_1 \triangleright S_3 \triangleright S_2 = R_2 \quad (3.22)$$

$$L_2 = S_2 \triangleright S_1 \triangleright S_3 \neq S_2 \triangleright S_3 \triangleright S_1 = R_3 \quad (3.23)$$

$$L_3 = S_3 \triangleright S_1 \triangleright S_2 \neq S_1 \triangleright S_3 \triangleright S_2 = R_2 \quad (3.24)$$

$$L_3 = S_3 \triangleright S_1 \triangleright S_2 \neq S_2 \triangleright S_3 \triangleright S_1 = R_3 \quad (3.25)$$

Formulas (3.22)-(3.25) are validated by the fact that the sequence operator $\triangleright$ is not commutative and the mutual exclusion operator $\otimes$ is commutative. Finally we have:

$$(S_1 \triangleright S_2 \triangleright S_3) \otimes (S_2 \triangleright S_1 \triangleright S_3) \otimes (S_3 \triangleright S_1 \triangleright S_2) \otimes (S_3 \triangleright S_2 \triangleright S_1) \neq$$

$$\neq (S_1 \triangleright S_2 \triangleright S_3) \otimes (S_1 \triangleright S_3 \triangleright S_2) \otimes (S_2 \triangleright S_3 \triangleright S_1) \otimes (S_3 \triangleright S_2 \triangleright S_1) \quad (3.26)$$

So we can conclude:

$$(S_1 \Leftrightarrow S_2) \Leftrightarrow S_3 \neq S_1 \Leftrightarrow (S_2 \Leftrightarrow S_3) \quad (3.27)$$

Formula (3.27) directly refutes our initial hypothesis (3.15). So, we just proved that the unordered sequence operator does not hold associativity property.

3.3.2.2 Parallelism with Communication

The parallelism operator with communication (also called *synchronization* operator) allows building a composite service out of two composite services. The services are executed concurrently as in case of the ordinary parallelism, but additionally they have mutually dependent operations which must be synchronized during the execution.

Definition 3.14. Let $C = \{(a,b) \mid (a,b) \in T_1 \times T_2 \cup T_2 \times T_1\}$ be a set of communication elements between services *S1* and *S2*, such that $(a_i, b_i) = \{p_i, (a_i, p_i), (p_i, b_i)\}, i = \overline{1,n} \in N$.

Definition 3.15. The composite service *S* being the parallelism with communication of two component services $S_1 \overset{C}{+} S_2$ is a tuple *S* = (*N, D, L, URL, CS, SN*) where:

- *N* is the name of the composite service;
- *D* is the composite service description;
- *L* is the location of the composite service;
- *URL* is the invocation of the composite service;
- $CS = CS_1 \cup CS_2$;
- $SN = (P, T, F, i, o)$ where:
  - $P = P_1 \cup P_2 \cup \{i, o\} \cup \{p_i \mid p_i \in (a_i, b_i) \in C, i = \overline{1,n}\}$;
  - $T = T_1 \cup T_2 \cup \{t_i, t_o\}$;
  - $F = F_1 \cup F_2 \cup \{(i, t_i), (t_i, i_1), (t_i, i_2), (o_1, t_o), (o_2, t_o), (t_o, o)\} \cup$
    $\cup \{(a_i, p_i), (p_i, b_i) \mid (a_i, b_i) \in C\}$

Figure 54 shows the graphical representation of the composite service $S = S_1 \overset{C}{+} S_2$.

The parallelism with communication construct is an advanced construct of the second type. It presents an enhancement to the basic parallelism construct extending it with an additional synchronization facility.

The parallelism operator with communication is used instead of the ordinary parallelism operator basically whenever two concurrent processes need to synchronize or exchange certain data during the execution. We already mentioned this application of parallelism within the section devoted to the parallelism operator, where we classified this use case as one for mutually dependent services. However, in the previous section devoted to the unordered sequence operator we

showed that this operator also suits for formalization of mutual dependence between two services. The difference is that while the unordered sequence operator formalizes a logical mutual dependence or complement, the parallelism operator with communication formalizes a functional dependence, which does not allow services being executed sequentially.

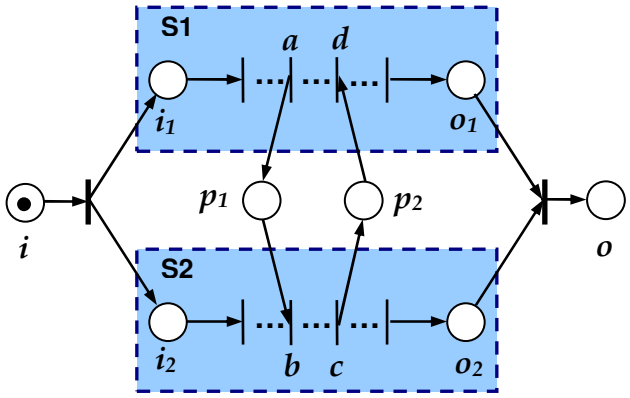

FIGURE 54 Parallelism with communication of two services

From the formal viewpoint, the composite service net is extended with communication elements defined by the set *C*. These elements are triples. Each triple is a place with one incoming and one outgoing arc. These triples are associated to pairs of transitions. Each such pair is bipartite, i.e. transitions within the pair belong to different component service nets and correspond to functionally correlated service operations. The triples of communication elements have univocal correspondence to the pairs of service operations. The first arc of a triple goes from the first transition of the pair to the place of the triple, while the second arc of the triple goes from the place of the triple to the second transition of the pair. If certain transition within one service net has an incoming communication arc, it cannot fire until its correlated transition within the other service net fires. In this way the execution of two concurrent services can be synchronized.

Let us return to our previous example of the travel purchase service (see Figure 47) and apply the parallelism operator with communication to build a more elaborate service. Figure 55 shows our modified version of the travel purchase service.

It is, perhaps, the simplest possible example of services' synchronization. When a customer books a ticket nobody actually knows in advance if a ticket will be available. But accommodation might be available and with a high certainty. However, it is obvious that there is no sense in reserving accommodation until it is clear that the tickets are available, especially taking into account the high availability of hotel services. So, it is logical that hotel should be reserved only after

the ticket's availability is guaranteed. The service shown in Figure 55 implements such dependence by applying the parallelism operator with communication for concurrent but synchronized execution of the ticket booking and hotel booking services.

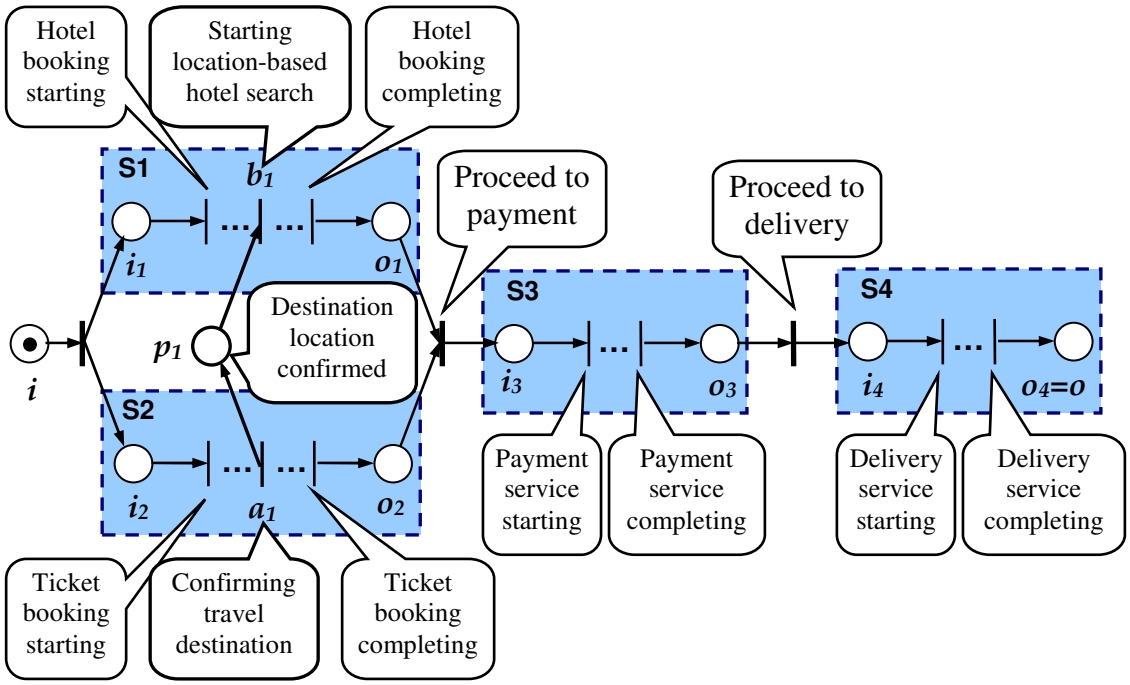

FIGURE 55 Travel purchase service with synchronized booking services

Another possible aspect is data exchange between two services. Though our service modeling framework does not functionally formalize such data exchange, it establishes special processing elements facilitating data sending and retrieval and formalizing them as process-related instances. The actual data exchange mechanisms have to be incorporated into synchronized services' core functionality. An example of synchronization based on data exchange can be also seen in Figure 55. In addition to a logical dependence of the hotel reservation service on the results produced by the ticket booking service, which, by the way, is not formally the necessary condition of synchronized and concurrent execution of these services, the hotel reservation service is also functionally dependent on the ticket booking service. The functional dependence lies in the hotel reservation service's need of knowing the destination of the user's travel in order to find an appropriate hotel at the desired location. The user's travel destination location can, in turn, be provided by the ticket booking service during its execution. So, it is reasonable to organize the component services execution as synchronized concurrency because the data required by one service can be produced by the other service earlier than its completion, at a certain point during execution, i.e., there is no need to wait until the completion of the first service. Secondly, the first service has time to perform

all its necessary operations preceding the synchronized operation prior to the readiness of the synchronized data.

However, the example presented in Figure 55 does not entail a strict necessity for the use of synchronization operator because two synchronized services have only a one-way dependence. Services having a one-way dependence can be easily composed by the sequence operator with, perhaps, somewhat reduced complexity comparing to the parallelism with communication operator. The real strict grounding for using the synchronization operator arises whenever the services are mutually dependent, i.e. none of them can complete without the other.

The same travel purchase service with the mutually dependent booking services is presented in Figure 56.

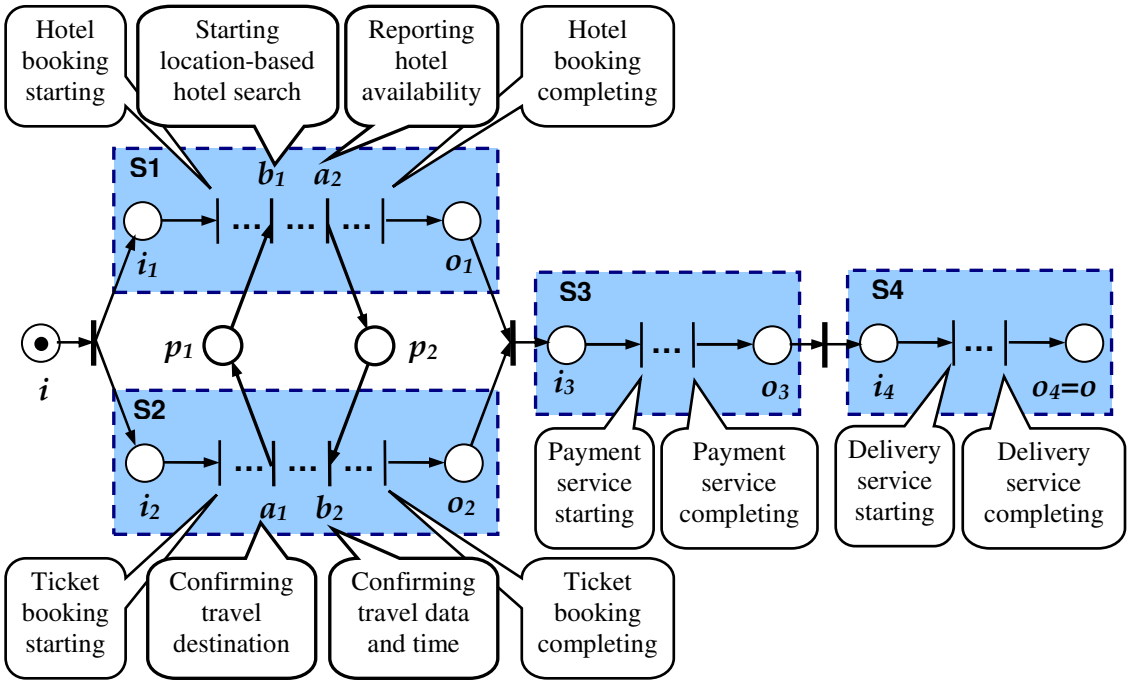

FIGURE 56 Travel purchase service with mutually dependent booking services

In the example shown in Figure 56 the booking services are mutually dependent. The hotel booking service requires information about the travel destination location during its execution. The ticket booking service in its turn needs information about hotel availability on the date of travel in order to complete tickets' reservation procedure. So, when tickets' availability is confirmed, the ticket booking service sends travel destination location data to the hotel reservation service. Then the hotel reservation service looks for an appropriate hotel accommodation starting on the date of travel and reports accommodation

availability back to the ticket booking service. Only after that the ticket reservation procedure can be completed.

Thus, when two services are mutually dependent, they cannot be organized by the sequence operator because none of the services can be completed prior to the other. Therefore, the use of the synchronization operator is absolutely necessary for this case.

As we can see, the parallelism operator with communication is formally very similar to the ordinary parallelism operator. Obviously, the parallelism operator with communication becomes equivalent to the parallelism operator when the communication set $C = \varnothing$. In other words, the parallelism operator $S_1 + S_2$ is a special case of the parallelism operator with communication $S_1 \overset{C}{+} S_2$, such that

$$S_1 + S_2 = S_1 \overset{\varnothing}{+} S_2 \tag{3.28}$$

From the definition of the parallelism operator, formula (3.28) and particularly from the way we define the communication set *C* we can conclude that the parallelism operator with communication holds the commutativity property:

$$S_1 \overset{C}{+} S_2 = S_2 \overset{C}{+} S_1 \tag{3.29}$$

As regards the associativity property, the synchronization operator encounters some complexities. These complexities arise from the binary nature of the communication sets as they are defined. When the communication sets are empty, we have the case of the ordinary parallelism operator, which is associative as shown by (3.9). So, extending the notation of the basic parallelism to the parallelism with the communication type of operator as shown by (3.28), we have:

$$\left(S_1 \overset{\varnothing}{+} S_2 \right) \overset{\varnothing}{+} S_3 = S_1 \overset{\varnothing}{+} \left(S_2 \overset{\varnothing}{+} S_3 \right) \tag{3.30}$$

Let us consider a special case where three services are linked by the parallel operator with communication. Figure 57 shows the corresponding service net.

In Figure 57 we have three parallel services and three communication sets:

$$C_{12} = \{(a_1, b_1), (a_2, b_2)\} \tag{3.31}$$

$$C_{13} = \{(a_3, b_3), (a_4, b_4)\} \tag{3.32}$$

$$C_{23} = \{(a_5, b_5), (a_6, b_6)\} \tag{3.33}$$

So, now we can build three composite services using three binary operators of parallelism with communication:

$$S_{12} = S_1 \overset{C_{12}}{+} S_2 \tag{3.34}$$

$$S_{13} = S_1 \overset{C_{13}}{+} S_3 \tag{3.35}$$

$$S_{23} = S_2 \overset{C_{23}}{+} S_3 \tag{3.36}$$

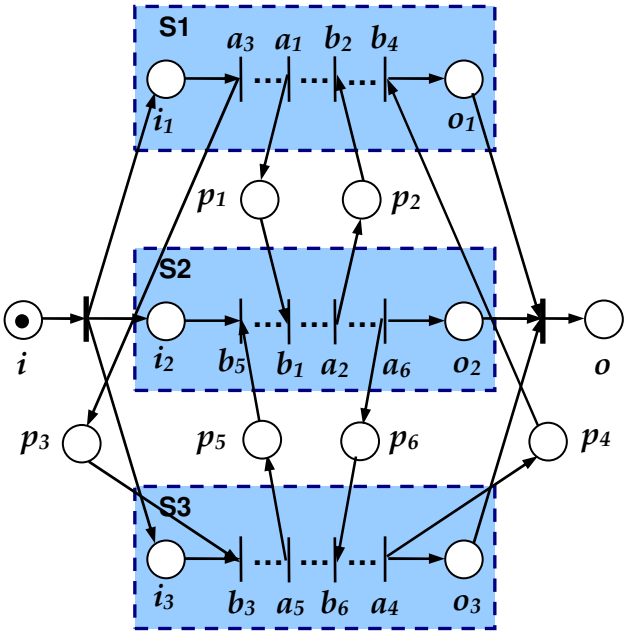

FIGURE 57 Synchronization of three services

We are, however, interested in building the composite service out of three component services using the parallelism operator with communication. This can be one of the following combinations:

$$S = S_{12} \overset{C_{123}}{+} S_3 = \left(S_1 \overset{C_{12}}{+} S_2 \right) \overset{C_{123}}{+} S_3 \tag{3.37}$$

$$S = S_{13} \overset{C_{132}}{+} S_2 = \left(S_1 \overset{C_{13}}{+} S_3 \right) \overset{C_{132}}{+} S_2 \tag{3.38}$$

$$S = S_{23} \overset{C_{231}}{+} S_1 = \left(S_2 \overset{C_{23}}{+} S_3 \right) \overset{C_{231}}{+} S_1 \tag{3.39}$$

From a structural viewpoint there is no problem in combining three services in parallel. But taking into account the specialty of the synchronization operator we still have to define the particular communication sets for connecting third services to the previously composed pairs of services. These sets are defined as follows:

$$C_{123} = C_{13} \cup C_{23} = \{(a_3, b_3), (a_4, b_4), (a_5, b_5), (a_6, b_6)\} \tag{3.40}$$

$$C_{132} = C_{12} \cup C_{23} = \{(a_1, b_1), (a_2, b_2), (a_5, b_5), (a_6, b_6)\} \tag{3.41}$$

$$C_{231} = C_{12} \cup C_{13} = \{(a_1, b_1), (a_2, b_2), (a_3, b_3), (a_4, b_4)\} \tag{3.42}$$

Now, we want to check if the associativity property holds for the parallelism operator with communication. Taking into account properties (3.37)-(3.39) and definitions (3.40)-(3.42), we can write down the following:

$$\left(S_1 \overset{C_{12}}{+} S_2\right) \overset{C_1}{+} S_3 = S_1 \overset{C_2}{+} \left(S_2 \overset{C_{23}}{+} S_3\right), \tag{3.43}$$

if and only if:

$$C_2 = C_{12} \cup C_{13} \tag{3.44}$$

and

$$C_{23} = C_1 - C_{13} \tag{3.45}$$

So, we see that the parallelism operator with communication does hold the associativity property. However, this is a structural associativity, not algebraic one, since we have differently defined communication sets on different sides of the equation (43).

In order to speak about algebraic or, even better, notational associativity, an additional assumption has to be made. We state that algebraic associativity takes place when two out of three services are independent of each other, i.e. their corresponding communication set is empty. Formally it can be presented as follows:

If

$$C_{13} = \varnothing, \tag{3.46}$$

then from (3.40)

$$C_{123} = C_{23} \tag{3.47}$$

and from (3.42)

$$C_{231} = C_{12} \tag{3.49}$$

Formulas (3.37) and (3.39) can be rewritten as follows:

$$S = S_{12} \overset{C_{123}}{+} S_3 = \left(S_1 \overset{C_{12}}{+} S_2\right) \overset{C_{23}}{+} S_3 \tag{3.50}$$

$$S = S_{23} \overset{C_{231}}{+} S_1 = \left(S_2 \overset{C_{23}}{+} S_3\right) \overset{C_{12}}{+} S_1 \tag{3.51}$$

Now, comparing the communication sets in formulas (3.50) and (3.51), and using the commutativity property (3.29) and the associativity property (3.30), we can conclude:

$$\left(S_1 \overset{C_{12}}{+} S_2\right) \overset{C_{23}}{+} S_3 = S_1 \overset{C_{12}}{+} \left(S_2 \overset{C_{23}}{+} S_3\right) \tag{3.52}$$

One more interesting aspect of the parallelism operator with communication is communication conflict.

Definition 3.16. *Communication conflict* or *communication lock* is a situation where two or more services are composed with the parallelism operator with communication in the way that at least one of them cannot be completed, i.e., the composite service cannot be completed. The common reason for communication locks emergence is incorrect organization of communication sets. Speaking more specifically, it is possible to say that, within any service, operations are executed in a certain order, so that one operation basically depends on another. Thus, an operation cannot be executed until its preceding operation is executed. Introduction of communication sets establishes extra and more sophisticated dependencies between services' operations. Simply speaking, we obtain a communication lock when the operation *c* depends on the operation *a* with respect to the service execution order, and the operation *a* depends on the operation *c* transitively through the operation *b* belonging to another service, and all three operations belong to the corresponding communication set. Figure 58 illustrates this simple example.

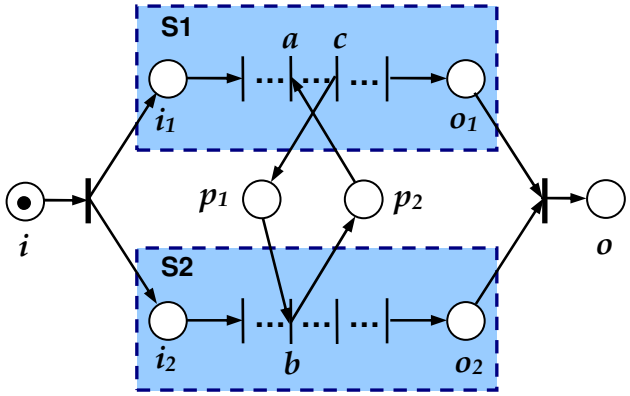

FIGURE 58 Simple communication lock

In Figure 58 we have a composition of two services with the parallelism operator with communication. The communication set of this composition is as follows:

$$C = \{(c,b),(b,a)\} \tag{3.53}$$

Everything seems fine with this communication set, but from the structure of the service S_1 we have that *a* precedes *c* within the order of the service execution, so we will end up with a communication conflict.

Formally speaking, operation pairs (*a*,*b*) within the communication set *C* are relations of the type "*a* precedes *b*". This relation is transitive by definition, i.e., if "*a* precedes *b*" and "*b* precedes *c*" then "*a* precedes *c*". We will use the notation $\mapsto$ instead of the word "precedes" from now on.

So far, a communication conflict can be defined as a unitary property $L_c(b_j)$ such that

$$L_c(b_j)=\begin{cases} 0, & if \begin{cases} a_i \mapsto b_j, a_i \in C, b_j \in C \\ \quad and \\ a_i \mapsto b_j, a_i \in S, b_j \in S \end{cases} \\ 1, & if \begin{cases} a_i \mapsto b_j, a_i \in C, b_j \in C \\ \quad and \\ b_j \mapsto a_i, a_i \in S, b_j \in S \end{cases} \end{cases} \tag{3.54}$$

where

- a_i and b_j are the service operations;
- *S* is the sequence of service operations;
- *C* is the communication set.

In terms of algebra of logics, a transitive relation $\mapsto$ is defined as follows:

$$\begin{cases} a \mapsto b \\ b \mapsto c \end{cases} \Rightarrow a \mapsto c \tag{3.55}$$

An intransitive relation $\mapsto$ is then defined in the following way:

$$\begin{cases} a \mapsto b \\ b \mapsto c \end{cases} \Rightarrow c \mapsto a \tag{3.56}$$

In other words, intransitive relation represents a cyclic dependence or a lock. Now, let us associate the pairs of the communication set with antecedents (left-hand side) of the expressions (3.55) and (3.56), and the operations execution order of the service with the consequent (right-hand side) of the corresponding expressions. For the case illustrated in Figure 58 we obtain:

$$\begin{cases} c \mapsto b \\ b \mapsto a \end{cases} \Rightarrow a \mapsto c \tag{3.57}$$

Apparently, the relation (3.57) is intransitive, and Figure 58 describes the communication lock. So, we can conclude that a service net is subjected to a

communication lock whenever the pairs of the communication set and the service execution order constitute the intransitive corollary relation between a pair of operations a_i, b_j.

$$L_c(b_j) = \begin{cases} 1, & if \begin{cases} a_i \mapsto b_i \mid (a_i, b_i) \in C \\ \quad ... \qquad\qquad \Rightarrow b_j \mapsto a_i \mid a_i, b_j \in S \\ a_j \mapsto b_j \mid (a_j, b_j) \in C \end{cases} \\ 0, & otherwise \end{cases} \tag{3.58}$$

In terms of Petri nets a communication lock is basically a deadlock. A deadlock is a transition or a set of transitions which cannot be fired. According to the Petri nets terminology, a transition is live if it is not deadlocked. However, liveness does not mean that a transition is enabled or can always be enabled. It rather implies that a transition is potentially fireable, i.e. there exists a marking in which the transition is enabled. A deadlock can still occur if the corresponding marking is not reached during the Petri net execution. Nevertheless, if such a marking exists, the deadlock is not absolute, only possible. An absolute deadlock occurs when the transition cannot fire in any reachable marking of the Petri net. In our case, we have an absolute type of deadlock due to the principle of service nets' design, which dictates the initial marking with only one token in the input place and the absence of cycles during execution. However, a presence of the absolute deadlock within the service net does not necessarily mean that the service cannot be completed. The principles of service nets' building still allow the presence of certain non-deterministic constructs. So, the final marking of the service net, which corresponds to the service's completion, and which dictates the only token within the whole net to be placed in the output place, potentially can be reachable through a different sequence of transition firings provided that none of the transitions constituting this sequence presents a communication lock.

Figure 59 illustrates the case when the service can still be competed in the presence of a communication lock.

As can be seen from Figure 59, the service S_1 contains an absolute communication lock *a*. Nevertheless, the services S_1 and S_2 can still be completed because they contain the mutual exclusion constructs branching the services' execution and preceding the communication lock. So, the service S_1 is potentially executable via the sequence of operations *defg* and the service S_2 is potentially executable via the sequence of operations *hjkl* so that ignoring the communication locks *a* and *b* are located within the alternative execution branches. This may be referred to as *avoidable* communication lock, while an *absolute* communication lock prevents the service from being completed in its presence. We will discuss deadlocks more thoroughly within the service composition analysis section.

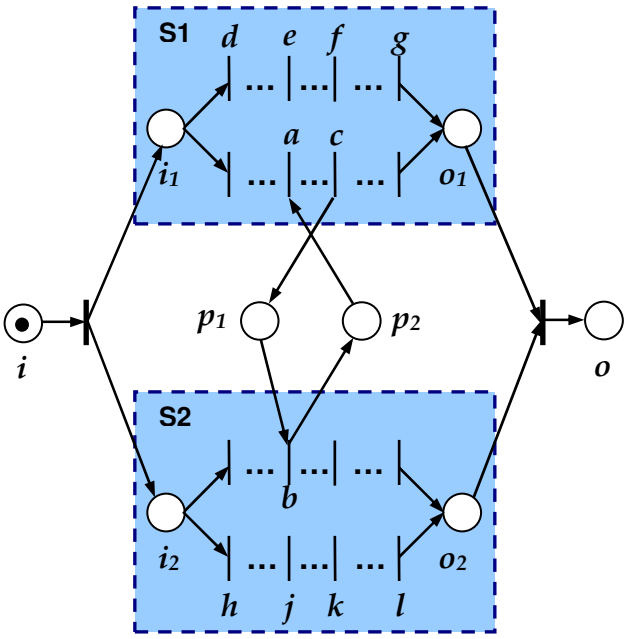

FIGURE 59 Avoidable communication lock

As we could see, detecting possible communication locks is, perhaps, a trivial problem when combining two services under the parallelism operator with communication. However, it is no longer that easy when the number of composed services becomes greater. There might arise dependencies which are much harder to detect due to the fact that every pair of composed services may have a separate communication set, and these communication sets can somehow overlap.

The worst possible scenario is when, say, three services S_1, S_2 and S_3 are composed, and their one-to-one communication sets C_{12}, C_{13} and C_{23} do not contain communication locks, however, their joint communication set $C = C_{12} \cup C_{13} \cup C_{23}$ contain such lock(s). These hidden communication locks are much harder to detect since the joint communication set C should be separately checked. Needless to say that such additional check along with the increase of the number of component services and communication elements between them does not facilitate preservation of scalability of the parallelism operator with communication. Moreover, the component communication sets C_{12}, C_{13} and C_{23} should be first checked with respect to the execution order of one-to-one service combinations. Then the joint communication set C should be checked in order to identify contradictory or intransitive dependencies.

Figure 60 shows an example of combining three services with the parallelism operator with communication. The hidden communication locks are indicated with thick red bars.

We can easily see from Figure 60 that the component communication sets:

$$C_{12} = \{(a_1, b_1), (a_2, b_2)\} \tag{3.59}$$

$$C_{13} = \{(a_3, b_3), (a_4, b_4)\} \tag{3.60}$$

$$C_{23} = \{(a_5, b_5), (a_6, b_6)\} \tag{3.61}$$

do not contain conflicts with respect to the services' operation execution order.

However, the joint communication set *C*:

$$C = C_{12} \cup C_{13} \cup C_{23} = \{(a_1, b_1), (a_2, b_2), (a_3, b_3), (a_4, b_4), (a_5, b_5), (a_6, b_6)\} \tag{3.63}$$

does contain communication locks.

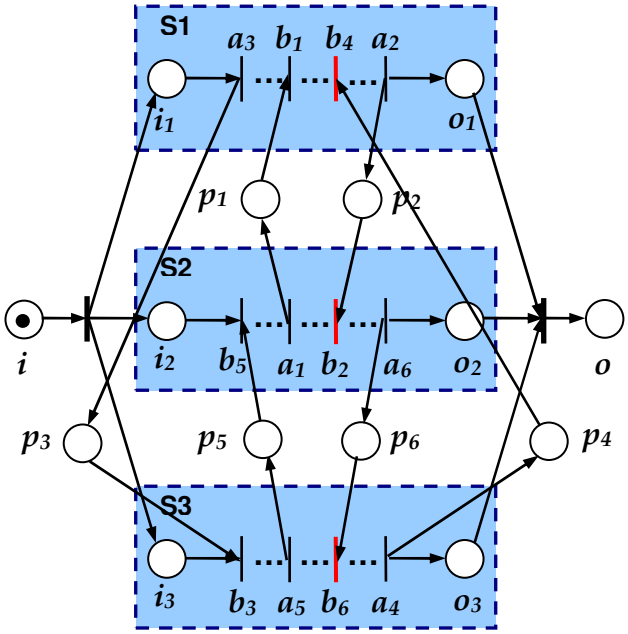

FIGURE 60 Hidden communication locks

Detecting communication locks in such a set with respect to the execution order of three different services is not a trivial task. We propose a special algorithm for communication locks detection within joint communication sets. We start with several assumptions:

1. Each communication place p_i has exactly one incoming and one outgoing arc, i.e., communication between services is purely deterministic.
2. A single service operation can be simultaneously input and output operation within different communication pairs, i.e. $a_i = b_j$.
3. Communication lock is always the output operation within any communication pair, i.e. the operation labeled as b_j.

Taking into account the presented assumption the detection algorithm operates as follows:

Step 1. Build the set E_i of operation pairs reflecting the order of operations' execution for every component service S_i. The set is defined as follows:

$$E_i = \{(o_j, o_k) \mid o_j \mapsto o_k; o_j, o_k \in S_i\} \tag{3.64}$$

Now let's build the corresponding sets for the component services from the example shown in Figure 60. If we encounter the situation described by the assumption 2, we add an extra pair (b_j, a_i) to the set E. The order of elements within the pair is strict and is justified by the fact that any operation, if it simultaneously participates in communication process as input and output operation, first acts as an output operation and then as an input operation. If there are operations in the service execution order which do not belong to the communication set C, we denote them as o_k.

$$E_1 = \{(a_3, b_1), (b_1, b_4), (b_4, a_2)\} \tag{3.65}$$

$$E_2 = \{(b_5, a_1), (a_1, b_2), (b_2, a_6)\} \tag{3.66}$$

$$E_2 = \{(b_3, a_5), (a_5, b_6), (b_6, a_4)\} \tag{3.67}$$

Step 2. Build the communication lock set L by serially scanning the joint communication set C and copying out all the operations labeled as b_j.

$$L = \bigcup_C \{b_j\} \mid b_j \in (a_j, b_j) \in C \tag{3.68}$$

For the example from Figure 60 the communication lock set is the following according to the formula (3.63):

$$L = \{b_1, b_2, b_3, b_4, b_5, b_6\} \tag{3.69}$$

Then create sets $T_1,\ldots,T_N$ and a joint set T such that:

$$T_j = \{\varnothing\}, j = \overline{1, N}, N = |L| \tag{3.70}$$

$$T = \bigcup_{i=1}^{N} T_j \tag{3.71}$$

Step 3. Set $j = 1$, $n = 1$.

Step 4. Take the element b_j of the set L and find the corresponding pairs (b_j, x_k) within all the sets E_i. Then if within found pairs $x_k = a_k$, add these pair to the set T_j:

$$T_j = T_j + \left\{ \bigcup_{i=1}^{S} \bigcup_{m=1}^{|E_i|} a_k \mid e_m = (b_j, a_k) \right\} \tag{3.72}$$

For our example the first element of the set L is b_1. The only pair of the type (b_1, x_k) is (b_1, b_4) within the set E_1. So, we go to Step 5.

Step 5. Now find a pair (b_j, x_k) such that $x_k \notin T_j$ within E_i. If no such pair can be found proceed to Step 7.

Step 6. If $x_k = b_k$ or $x_k = o_k$, find next pair of the type (b_k, x_l) or (o_k, x_l) within the same set E_i. If found pair is again of the type (b_k, b_l) or (b_k, o_l) repeat Step 5 substituting k with l until the pair of the type (b_k, a_l) or (o_k, a_l) is found. If $a_l \notin T_j$, update the set T_j:

$$T_j = T_j + \{a_l\} \tag{3.73}$$

and repeat Step 5.

If no more pairs of the type (b_k, a_l) or (t_k, a_l) can be found within the set E_i, proceed to Step 7.

For our example the algorithm find next pair (b_4, a_2) within E_1. This is a successful pair since a_2 is not in the set T_1 and the algorithm updates the set T_1, so we have $T_1 = \{a_2\}$.

Step 7. If $n \le |T_j|$ take the element $t_n{}^j = a_k \mid t_n \in T_j$, increment $n = n + 1$ and go to Step 4 provided that $b_j = a_k$.

Otherwise, proceed to Step 8.

Step 8. If $j < |L|$, increment $j = j + 1$, set $n = 1$ and go to Step 4. Otherwise, go to Step 9.

Step 9. Set $j = 1$.

Step 10. Set $n = 1$.

Step 11. Now, when corollary sets T_j are built for every potential communication lock b_j, it is time to check which of the potential locks are real locks.

Take $t_n{}^j = a_k \mid t_n \in T_j$ and build new set B_k as follows:

$$B_k = \bigcup_{C} \{b_p\} \mid b_p \in (a_p, b_p) \in C, a_p = a_p \tag{3.74}$$

Set $p = 1$.

Step 12. If $T_p \neq \varnothing$, set $r = 1$ and proceed to Step 13. Otherwise, proceed to Step 15.

Step 13. Take $t_r{}^p = a_s \mid t_r \in T_p$ and find a pair $(a_s, b_j) \in C$. If such pair can be found, then b_j is a communication lock. Then if $j < |L|$ increment $j = j + 1$ and go to Step 10. Otherwise, go to Step 18.

Otherwise, repeat Steps 11 – 13 provided that $a_k = a_s$.

Step 14. If $r < |T_p|$ increment $r = r + 1$ and repeat Step 13.

Step 15. If $p < |B_k|$ increment $p = p + 1$ and go to Step 12.

Step 16. If $n < |T_j|$ increment $n = n + 1$ and go to Step 11.

Step 17. b_j is not a communication lock. Remove b_j from the set L:

$$L = L - \{b_j\} \tag{3.75}$$

If $j < |L|$ increment $j = j + 1$ and go to Step 10.

Step 18. Return L.

As the result of the algorithm work we receive the set L containing all communication locks of the composite service.

The algorithm is capable of detecting not only ordinary communication locks, but also hidden communication locks, i.e., the locks containing nested transitive dependencies such as shown in Figure 60. For detection of ordinary communication locks it might be reasonable to use a more simple type of algorithm, which can be obtained as a reduction of our complex algorithm and which we do not present here.

3.3.2.3 Refinement

The refinement (or inclusion) operator allows building a composite service out of two component services. The first service is refined by including a second service into the execution process of the first service instead of its component operation or a sequence of operations. So, the execution of component services is nested: first the execution of the refined service starts, then at a certain point of it the execution of the included service starts, after that the execution of the included service ends, and finally the execution of the refined service ends.

Definition 3.17. The composite service S being the refinement of a component service S_1 with a component service S_2 denoted as $S = S_1 \overset{a}{\nabla} S_2$ is a tuple $S = (N, D, L, URL, CS, SN)$ where:

- N is the name of the composite service;
- D is the composite service description;
- L is the location of the composite service;
- URL is the invocation of the composite service;
- $CS = CS_1 \cup CS_2$;
- $SN = (P, T, F, i, o)$ where:
  - $P = (P_1 - \{i_a, o_a\}) \cup P_2 \mid i_a = i_2, o_a = o_2; i_a, o_a \in P_1; i_2, o_2 \in P_2$;
  - $T = T_1 \cup T_2$;
  - $F = (F_1 - \{(t_{i1}, i_a), (o_a, t_{o1})\}) \cup F_2 \cup \{(t_{i1}, i_2), (o_2, t_{o1})\}$;
  - $i = i_1$;
  - $o = o_1$.

Figure 61 shows the graphical representation of the composite service $S = S_1 \overset{a}{\nabla} S_2$.

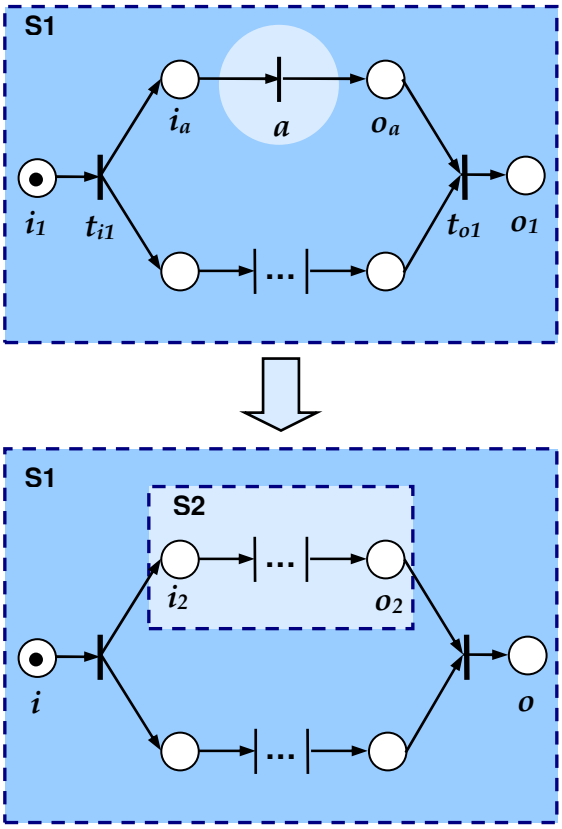

FIGURE 61 Refinement of service

The refinement construct is an advanced construct of the third type, i.e. a specific standalone construct for modeling of special cases.

However, despite its specifics, the refinement operator appears vitally important for modeling of composite services. Hierarchical modeling of sophisticated composite service seems hardly possible without the refinement operator.

What is, perhaps, most specific and interesting regarding this operator, is that it is the only construct which allows building partially composite services, i.e., composite services that present their own exclusive functionality. All other composition constructs described above in this Chapter allow, in contrast, building only pure composite services, i.e., composite services that do not contain their own functionality, being simply containers for component services. The refinement operator may be successfully used in the process of building services of both types though.

Thus, we see two main use cases for the refinement operator. The first one is actually the refinement function. The refinement operator can be successfully applied for enhancement of certain service quality by substituting some of its functionality with the same but higher quality functionality offered by a separate service. So, the operator of refinement just inserts this separate service into the refined service instead of the operation or the sequence of operations performing the corresponding refined functionality.

Figure 62 shows an example of the first use case of refinement operator.

Recalling our previous examples with the travel purchase composite service we may assume that the location-based hotel search functionality provided by the component hotel reservation service is not satisfactory for certain cases. However, we can find a separate location-based service which performs location search with a significantly greater accuracy, say, comparable to GPS accuracy. What do we do then? We apply the refinement operator over the set of operations a within our hotel reservation service S_1 and insert in their place a new service S_2, which performs a location-based search functionality similar to the functionality performed by the removed set of operations. As a result we have a sort of nested services S_1 and S_2, which produce improved value for a customer of the hotel reservation service. The corresponding service nets of the hotel reservation service and the travel purchase service are illustrated in Figure 62. As we can observe, after the application of the refinement operator, the hotel reservation service becomes partially composite.

Another use case of the refinement composition operator is hierarchical composite service modeling and composite service modeling from scratch. Imagine that we do not have a travel purchase service and we want to create one. However, we do not want to design its functionality from scratch as well. Conversely, we would like to utilize existing services to incorporate different pieces of necessary functionality into our future travel purchase service. Then a sample structure or a service plan should be created. This plan (see Figure 63) basically represents the structure of the future service with the distinction that its component operations

are generic operations, i.e., they are labeled but not filled with functionality. Such plan may be called a *generic service*. As soon as the generic service is designed, for each its component operation a service should be found to perform the corresponding functionality. Then the operator of refinement is to be applied to insert found services in the place of component operations of the generic service. By doing that the composite service is built from scratch. The corresponding process is illustrated in Figure 63.

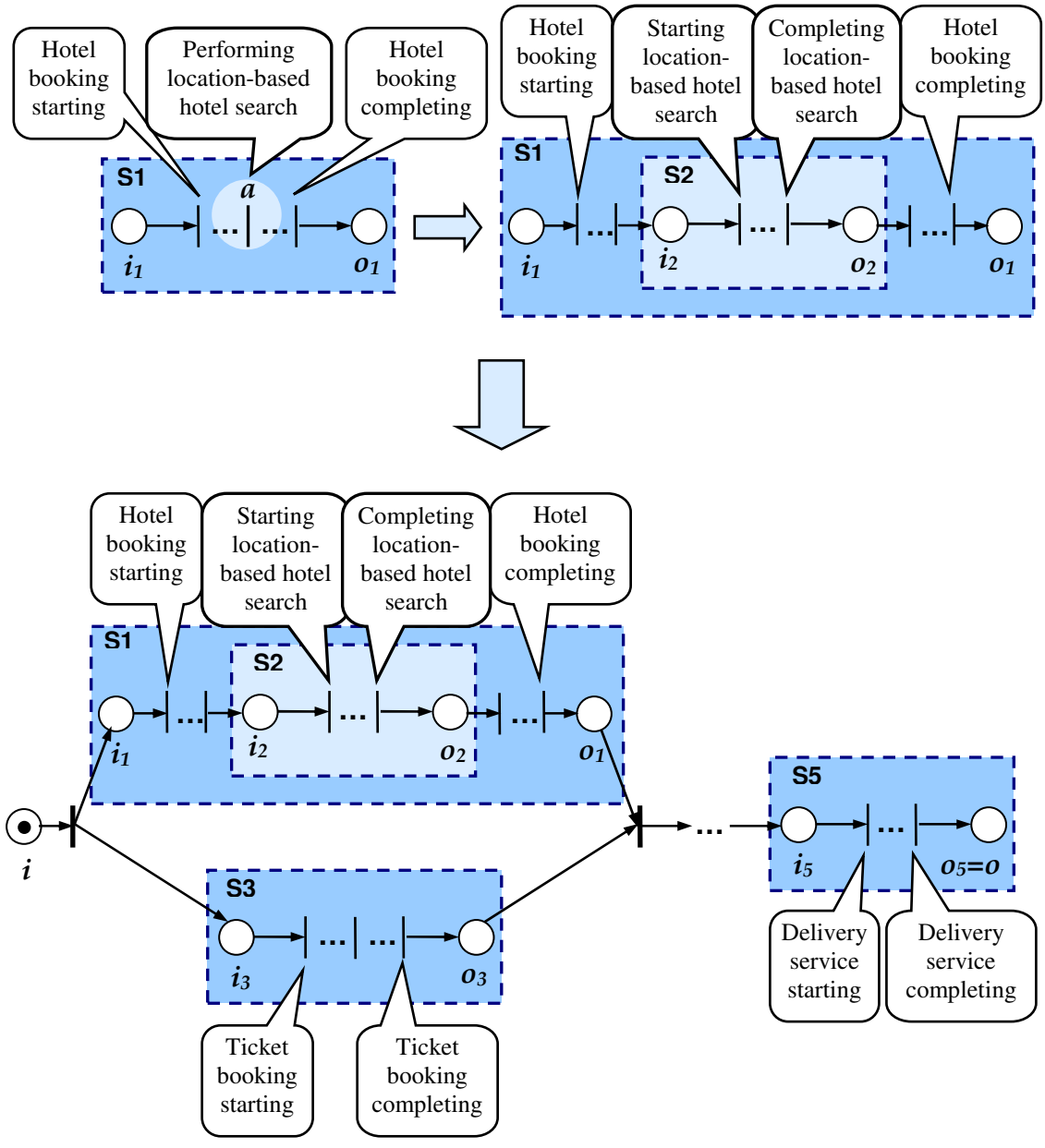

FIGURE 62 Travel purchase service with refined hotel reservation service

A slightly different service composition procedure takes place with hierarchical composite service modeling. Let the travel purchase service again be used as the example to explain how hierarchical composition is performed. First a generic service is built similarly to the one shown in Figure 63. Then this service is populated with real component services substituting its generic operations with real functionality. After that the travel booking service can be refined with two more component services performing separately ticket booking and hotel reservation procedures. These two services can be included into the travel booking service using the parallelism operator.

So, hierarchical composition process is basically the process of gradual refinement of component services with other component services. It can be particularly useful for enhancing existing composite services with superior quality and additional features. The corresponding process of the hierarchical refinement of the travel purchase service is illustrated in Figure 64.

Thus, we can now distinguish between two main approaches to service composition through planning. By service composition planning we mean composition modeling and model verification prior to actual composition implementation. One approach is preliminary composition planning. It is realized via composite service building from scratch (see Figure 63). The main idea is that a service is thoroughly planned and modeled in advance to its actual implementation. When applying this type of approach, purely composite services are usually built.

An alternative approach extends and restructures an existing composite service, i.e., it is service re-planning on the fly. This approach is useful when the needed service is already available but does not satisfy certain established quality requirements or lacks some desirable features. These quality and features can then be added to the existing service by gradually refining it and its component services until the required quality level is achieved. When applying this approach, partially composite services are often obtained.

As regards algebraic properties of the refinement operator, we can say that it is apparently not commutative:

$$S_1 \overset{a}{\nabla} S_2 \neq S_2 \overset{a}{\nabla} S_1 \tag{3.76}$$

since the inclusion of S_2 in S_2 is not functionally equivalent to the inclusion of S_1 into S_2. Moreover, inclusions on different sides of the inequality (3.76) cannot be both done over operation a, since such operation or set of operations basically belongs to only one of the services.

However, the refinement operator is associative by construction. Two possible orders of performing two nesting operations with three services apparently produce the same result if no other operations are performed on these services between two consequent nestings.

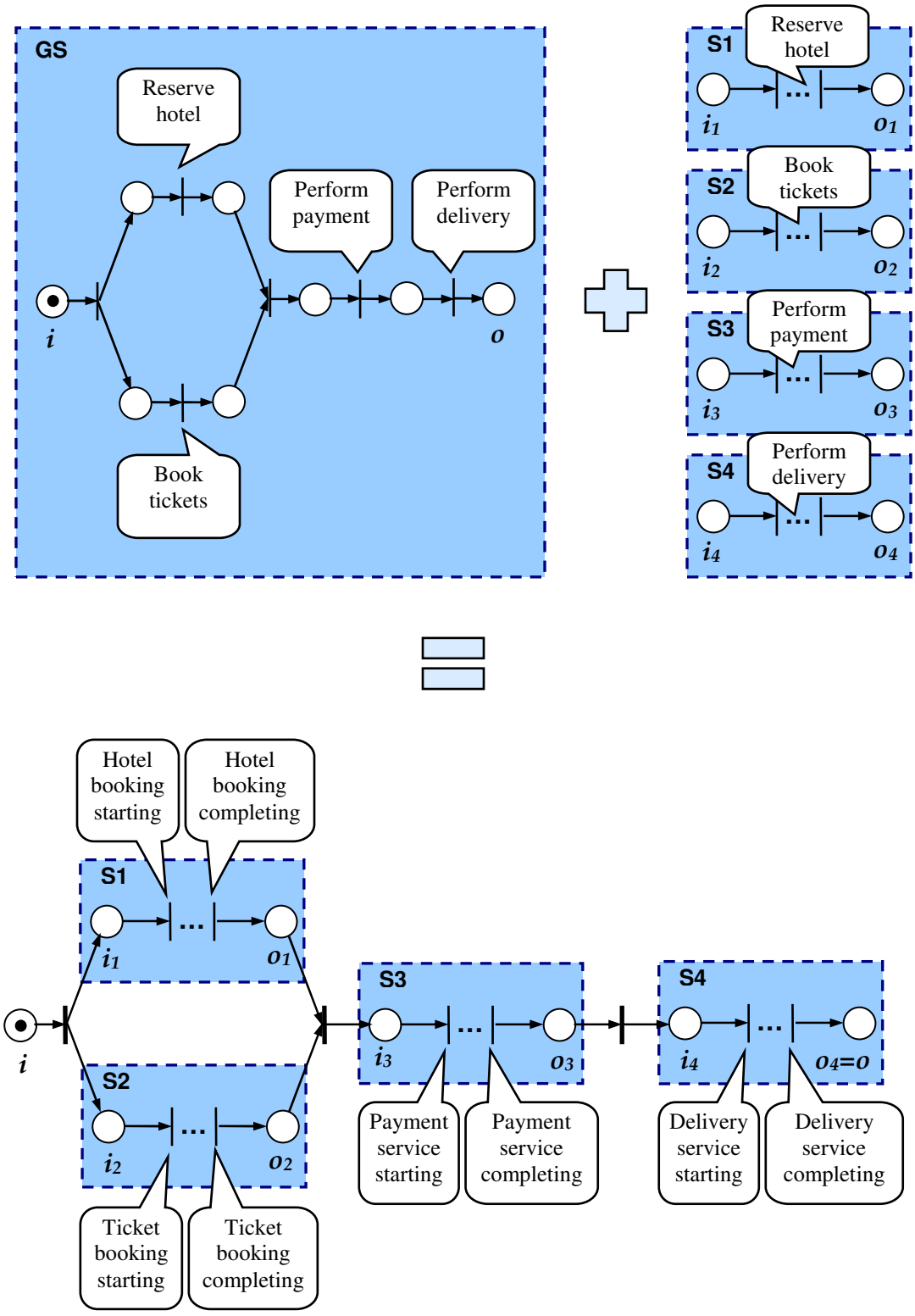

FIGURE 63 Composite service modeling from scratch using the refinement operator

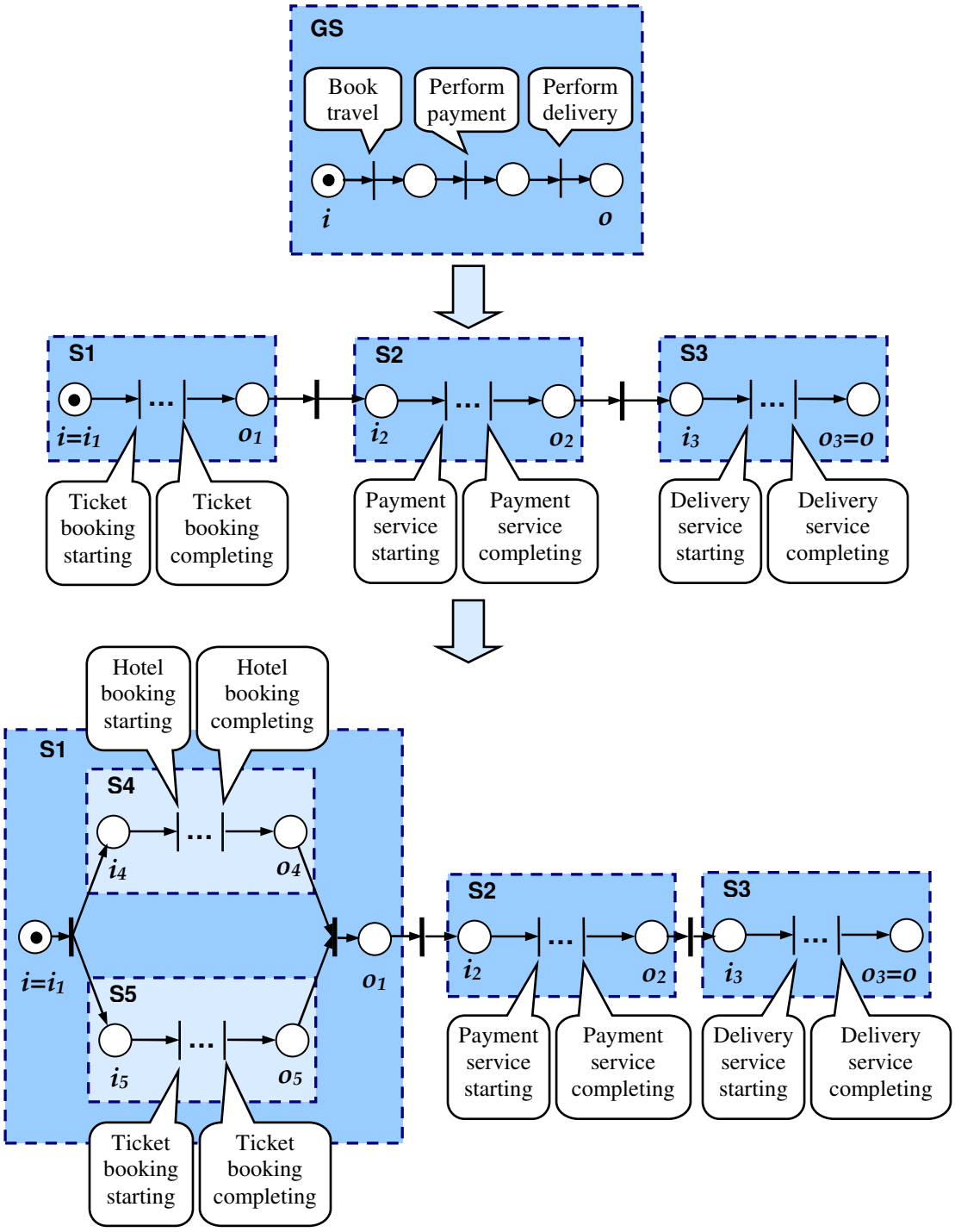

FIGURE 64 Hierarchical composition of a travel purchase service

$$\left(S_1 \overset{a}{\nabla} S_2\right) \overset{b}{\nabla} S_3 = S_1 \overset{a}{\nabla} \left(S_2 \overset{b}{\nabla} S_3\right) \tag{3.77}$$

3.3.2.4 Algebraic Properties

Before we speak about observed algebraic properties of the proposed algebraic operators and their combinations and before we can complete the description of our service algebra, we should first consider a complex composite service example, which would help us see how our service algebra operates in sophisticated service composition scenarios. This example should also illustrate how different composition operators can be jointly applied to get the desired result.

We again take as illustration the travel purchase service, which was given a variety of exemplary formalizations in the process of service constructs' description, some of which we reuse in the following example.

Figure 65 shows the final version of the composite travel purchase service. It can be easily seen that our composition uses a variety of composition operators in order to provide complex value-added functionality to service consumers. We have six component services in total to be included in our composite service. Service S_1 refers to hotel reservation operations, S_2 performs location-based search, S_3 is a ticket booking service, S_4 allows making payments via bank transfers, S_5 performs payments via Pay Pal system, and finally S_5 performs travel papers delivery to a final customer. We believe that this set of services constitutes necessary and, in many cases, sufficient functionality for providing travelers high-quality value-added autonomous travel purchase service. However, our main concern is not how to pick appropriate component services, but how to compose available component services in order to get the desired result.

Considering the operation process of our tentative service we can distinguish between three main operational stages which demonstrate logical causality. These three stages are booking, payment and delivery in a sequential order. The payment process logically depends on the booking process and can be started only after successful completion of all booking operations. The same is true for the delivery process that depends on completion of payment operations. Thus, it is obvious that the corresponding services - let S_b, S_p and S_d denote the booking, payment and delivery component services respectively - can be combined with two binary sequence operators as shown by formula (3.78).

$$S = S_b \triangleright S_p \triangleright S_d \tag{3.78}$$

Note that the order of the operands within the formula (3.78) is fixed since the sequence operator is not commutative.

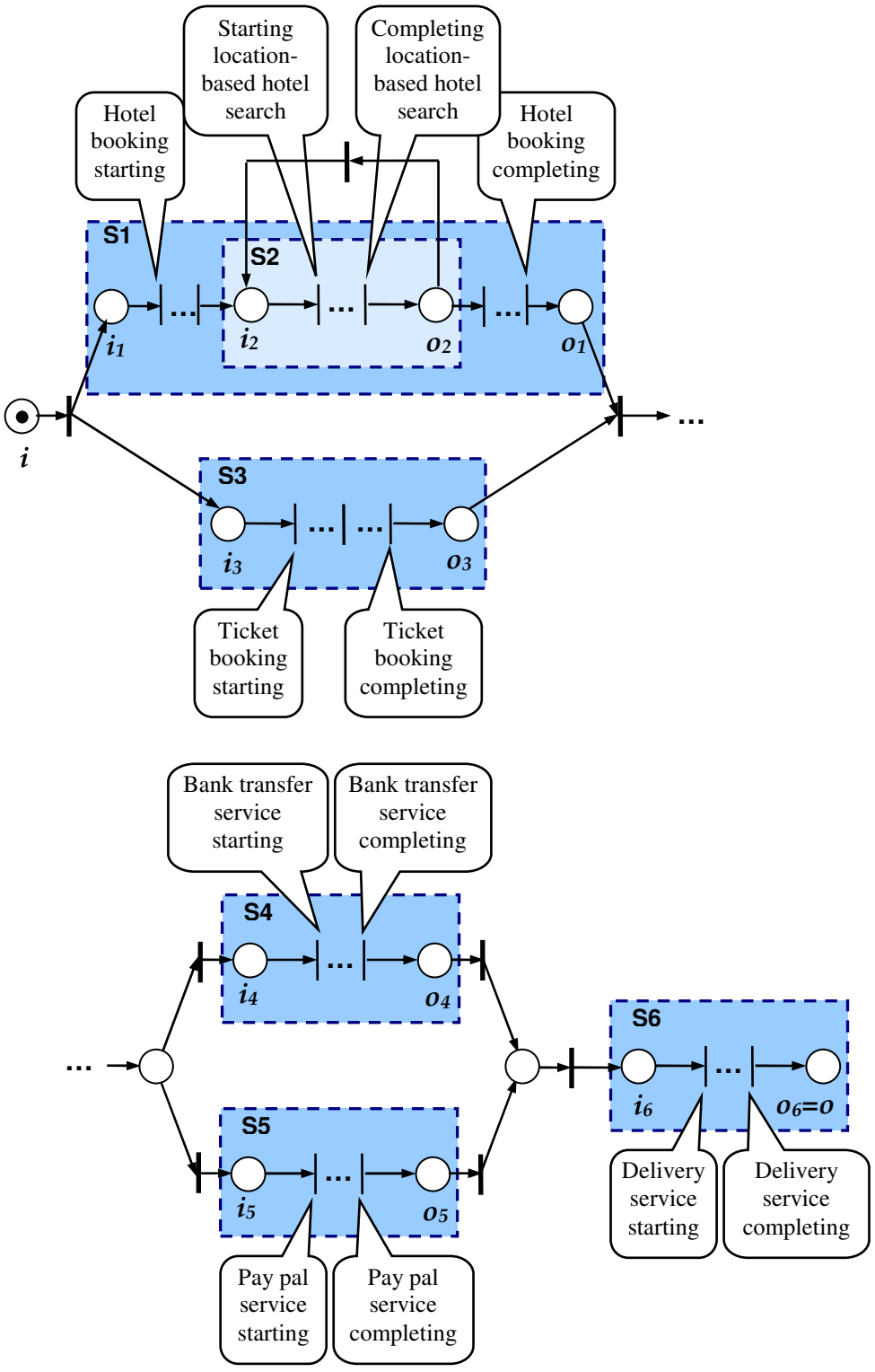

FIGURE 65 Travel purchase service: final composition

Now, let us consider the booking component service S_b. This service must perform two important operations: hotel reservation and ticket booking. These two operations can be in principle accomplished independently, so, they can be performed either in sequence or in parallel. The choice of the composition operator here is arbitrary or may be justified by certain functional or performance-related reasons. We choose the parallelism operator to compose hotel reservation and ticket booking services denoted as S_{hr} and S_{tb} respectively. The major reason justifying such a choice is that the two services might need to communicate during the execution, which is impossible in the case of sequential execution. For the sake of simplicity we, however, use in our example the ordinary parallelism operator instead of the parallelism operator with communication since they are algebraically equivalent until the number of the combined services exceeds 2. So, we have:

$$S_b = S_{hr} + S_{tb} \tag{3.79}$$

The hotel reservation service $S_{hr} = S_1$ does not, however, satisfy certain quality requirements concerning location-based hotel search. Nevertheless, there is a separate location-based search service S_2 that meets the established requirements. It is then logical to refine the hotel reservation service S_1 with the service S_2 using the refinement operator:

$$S_{hr} = S_1 \overset{a}{\nabla} S_2, \tag{3.80}$$

where a is the set of location-based hotel search operations such that $a \in S_1$.

Now, when the location-based hotel search produces satisfactory results, the number of the produced results should be considered. In the current formalization the hotel reservation service is capable of finding only one hotel based on the required location. Certainly, it is questionable whether the first hotel found satisfies customer's demands. It would be stupid to accommodate the user in an arbitrarily chosen hotel even if it meets his demands. A much more effective solution is to choose among several alternatives matching them with customer's demands. Even if we assume that the hotel reservation service is capable of doing such matchmaking, though it is out of our modeling scope, the location-based hotel search must first produce multiple alternatives for grounded hotel selection. If the location-based search service S_2 is incapable of producing multiple results with one pass, it should be executed multiple times, which is already a composition problem. So, we apply the iteration operator to the location-based search service in order to produce multiple hotel alternatives:

$$S_{hr} = S_1 \overset{a}{\nabla} S_{lbs} \tag{3.81}$$

$$S_{lbs} = \alpha S_2 \tag{3.82}$$

Provided that $S_{tb} = S_3$ and using formulae (3.79)-(3.82) we can now complete the booking component service:

$$S_b = S_1 \overset{a}{\nabla} \alpha S_2 + S_3 \tag{3.83}$$

As regards the payment service it would be useful to provide alternative payment methods within the composite travel purchase service. Fortunately, there are available two component services S_4 and S_5 implementing two different payment methods. These services must be executed alternatively to perform the travel payment with one method or the other. So, we apply the mutual exclusion operator to build the payment service out of two alternative services:

$$S_p = S_4 \otimes S_5 \tag{3.84}$$

Provided that $S_d = S_6$ and using formulae (3.78), (3.83) and (3.84) we can now complete the composite travel purchase service:

$$S = S_1 \overset{a}{\nabla} \alpha S_2 + S_3 \triangleright S_4 \otimes S_5 \triangleright S_6 \tag{3.85}$$

Formula (3.85) gives the algebraic representation of the exemplary travel purchase service, the Petri net representation of which is shown in Figure 65.

Although the algebraic expression (3.85) correctly represents the structure of the travel purchase service, it is still questionable whether it correctly represents the execution order of the service. Since the formula (3.85) includes no parentheses, it can be rewritten as follows:

$$S = \left(\left(\left(\left(S_1 \overset{a}{\nabla} (\alpha S_2)\right) + S_3\right) \triangleright S_4\right) \otimes S_5\right) \triangleright S_6 \tag{3.86}$$

Obviously, the formula (3.86) does not reflect the correct order of the services' execution in Figure 65. One way to cope this problem is to place the appropriate parentheses within the formula (3.85). The correct formula then is the following:

$$S = S_1 \overset{a}{\nabla} (\alpha S_2) + S_3 \triangleright (S_4 \otimes S_5) \triangleright S_6 \tag{3.87}$$

However, ordering algebraic operations only by their notational order and parentheses may adversely increase the notational complexity of the algebra because in many cases the number of necessary parentheses will be excessive.

To obtain a more concise and easily readable notation we have to introduce priority system linking a certain priority level to every algebraic operator. Then the operators within a certain algebraic expression will be executed in the order of their priorities. If, however, it is necessary that a less prioritized operator is executed prior to a more-prioritized, this operator with its operands is parenthesized, i.e., enclosed in parentheses.

Introduction of priority systems to algebras containing binary and higher arity operators is a common practice. Priority systems make algebraic notations more concise consequently increasing the notational power and efficiency of the algebras in question. However, the crucial point of priority systems is their design maturity. The main criterion is that priorities have to be assigned to operators to minimize the amount of parenthesizing in most commonly emerging cases. In other words, the prioritization should reflect the natural order of things that is being modeled by the corresponding algebra.

We propose the following priority system for our Web service algebra (the operators are listed in the order of priority degradation):

1. $\alpha, \overset{a}{\nabla}$ - iteration and refinement operators;
2. $\triangleright, \Leftrightarrow$ - sequence and unordered sequence operators;
3. $+, \overset{C}{+}$ - parallelism and parallelism with communication operators;
4. $\otimes$ - mutual exclusion operator.

Besides common sense, our main idea when designing the proposed priority system was that those operators which are more often applied to single component services should be more prioritized while those operators which are more frequently applied to sets of component services and mostly perform branching should be less prioritized. In this way the amount of parenthesizing can be significantly reduced.

A somewhat special case is the iteration operator. This unitary operator is mostly applied over a single service and should be, perhaps, even more prioritized than the refinement operator. Nevertheless, we put these two operators under the same priority because in cases where the iteration operator precedes the refinement operator it is in any case executed first due to the notational order as shown by expression (3.88).

$$\alpha S_1 \overset{a}{\nabla} S_2 = (\alpha S_1) \overset{a}{\nabla} S_2 \tag{3.88}$$

As for the case when the refinement operator precedes the iteration operator, the latter one is executed first due to its unitary nature:

$$S_1 \overset{a}{\nabla} \alpha S_2 = S_1 \overset{a}{\nabla} (\alpha S_2) \tag{3.89}$$

Now, when we have assigned priorities to our algebraic operators, we can rewrite the formula (3.85) taking into account the priorities:

$$S = \left(S_1 \overset{a}{\nabla} \alpha S_2 + S_3 \right) \triangleright (S_4 \otimes S_5) \triangleright S_6 \tag{3.90}$$

Though we have now even more parentheses than in formula (3.87), notational clarity has increased: the correspondence between the algebraic notation and the composite service structure is now more clearly noticeable.

So, algebraic expressions similar to (3.90) and built using the discussed algebraic operators and the priority system can uniquely and unambiguously describe any composite service. Thus, within the algebraic service model, service operations are represented by the operators, and component services by the operands; the order of the services' execution is determined by the operators, and the order of operations' execution is determined by the priority system and parentheses.

While Petri net models of Web Services are useful for graphical modeling, analysis and verification of services, our Web Service algebra is useful for modeling composite service structures, modifying those structures and easily mapping them into lower level Petri net representation. The designed Web service algebra presents a concise notation which is quite helpful for performing operations over services' structure. So, the Petri nets formalism is the basis of the composite services' modeling and the Web service algebra is the superstructure on top of the basis which allows manipulating objects on the composition layer in a clearer and more efficient way.

Algebraic properties, some of which we have already observed within the sections devoted to individual composition operators, are especially useful in the sense that they represent rules for manipulating composition objects and structure of composite services in a clear and formal way.

Below we list the algebraic properties of the described composition operators classifying them into several groups.

- Identity

$$S \triangleright S_0 = S \tag{3.91}$$

$$S_0 \triangleright S = S \tag{3.92}$$

$$S + S_0 = S \tag{3.93}$$

$$S \Leftrightarrow S_0 = S \tag{3.94}$$

$$S \overset{a}{\nabla} S_0 = S \tag{3.95}$$

- Idempotence

$$S \otimes S = S \tag{3.96}$$

$$\alpha S_0 = S_0 \tag{3.97}$$

- Commutativity

$$S_1 \otimes S_2 = S_2 \otimes S_1 \tag{3.98}$$

$$S_1 + S_2 = S_2 + S_1 \tag{3.99}$$

$$S_1 \Leftrightarrow S_2 = S_2 \Leftrightarrow S_1 \tag{3.100}$$

$$S_1 \overset{C}{+} S_2 = S_2 \overset{C}{+} S_1 \tag{3.101}$$

- Associativity

$$(S_1 \triangleright S_2) \triangleright S_3 = S_1 \triangleright (S_2 \triangleright S_3) \tag{3.102}$$

$$(S_1 \otimes S_2) \otimes S_3 = S_1 \otimes (S_2 \otimes S_3) \tag{3.103}$$

$$(S_1 + S_2) + S_3 = S_1 + (S_2 + S_3) \tag{3.104}$$

$$\left(S_1 \overset{C_1}{+} S_2\right) \overset{C_2}{+} S_3 = S_1 \overset{C_1}{+} \left(S_2 \overset{C_2}{+} S_3\right), C_1 \cap C_2 = \varnothing \tag{3.105}$$

$$\left(S_1 \overset{a}{\nabla} S_2\right) \overset{b}{\nabla} S_3 = S_1 \overset{a}{\nabla} \left(S_2 \overset{b}{\nabla} S_3\right), a \in S_1, b \in S_2 \tag{3.106}$$

- Distributivity

$$S_1 \triangleright (S_2 \otimes S_3) = (S_1 \triangleright S_2) \otimes (S_1 \triangleright S_3) \tag{3.107}$$

$$(S_1 \otimes S_2) \triangleright S_3 = (S_1 \triangleright S_3) \otimes (S_2 \triangleright S_3) \tag{3.108}$$

$$S_1 + (S_2 \otimes S_3) = (S_1 + S_2) \otimes (S_1 + S_3) \tag{3.109}$$

$$(S_1 \otimes S_2) + S_3 = (S_1 + S_3) \otimes (S_2 + S_3) \tag{3.110}$$

$$S_1 \Leftrightarrow (S_2 \otimes S_3) = (S_1 \Leftrightarrow S_2) \otimes (S_1 \Leftrightarrow S_3) \tag{3.111}$$

$$(S_1 \otimes S_2) \Leftrightarrow S_3 = (S_1 \Leftrightarrow S_3) \otimes (S_2 \Leftrightarrow S_3) \tag{3.112}$$

$$S_1 \overset{a}{\nabla} (S_2 \otimes S_3) = \left(S_1 \overset{a}{\nabla} S_2\right) \otimes \left(S_1 \overset{a}{\nabla} S_3\right) \tag{3.113}$$

$$(S_1 \otimes S_2) \overset{a}{\nabla} S_3 \neq \left(S_1 \overset{a}{\nabla} S_3\right) \otimes \left(S_2 \overset{a}{\nabla} S_3\right) \tag{3.114}$$

$$S_1 \triangleright (S_2 + S_3) = (S_1 \triangleright S_2) + (S_1 \triangleright S_3) \tag{3.115}$$

$$(S_1 + S_2) \triangleright S_3 = (S_1 \triangleright S_3) + (S_2 \triangleright S_3) \tag{3.116}$$

$$\alpha\left(S_1 \overset{C}{+} S_2\right) = (\alpha S_1) \overset{C}{+} (\alpha S_2) \tag{3.117}$$

- Other properties

$$S_1 \Leftrightarrow S_2 = (S_1 \triangleright S_2) \otimes (S_2 \triangleright S_1) \tag{3.118}$$

$$S \Leftrightarrow S = S \triangleright S \tag{3.119}$$

$$S_1 \overset{\varnothing}{+} S_2 = S_1 + S_2 \tag{3.120}$$

$$S_1 \overset{a}{\nabla} S_2 = S_1, if \quad a = \{\varnothing\}, \quad or \quad a \notin S_1 \tag{3.121}$$

Now let us give explanations to some of the properties and proofs where needed. Properties (3.91)-(3.93) are straightforward from the definition of empty service and the corresponding operators' definitions. Empty service located in the scope of the sequence or parallelism operator acts as an echo service, doing nothing but allowing further execution of the composite service. Separate properties (3.91) and (3.92) are established because the sequence operator is not commutative.

The parallelism with communication operator also satisfies an identity property. However, when an empty service is subjected to this operator the communication set is necessarily empty, so the parallelism operator with communication is reduced to the ordinary parallelism operator with respect to property (3.120) and possesses the same identity property (3.93) as the ordinary parallelism does.

Unordered sequence identity property (3.94) is a bit more difficult to verify. It is obtained by consecutive application of the properties (3.116), (3.91), (3.92) and (3.96).

Refinement identity property (3.95) is introduced intentionally to avoid possible ambiguity. Of course, algebraically nothing prohibits refining certain service operation by empty service, but this may occasionally turn the service inoperative, which is highly undesirable due to impossibility of formal detection. That is why we force the rule declaring that refinement with empty service does nothing.

Mutual exclusion idempotence property (3.96) is straightforward from the definition of the mutual exclusion operator. It is logical that the choice between two identical variables is identical to the variable.

Property (3.97) is justified by the fact that iteration of empty service, similarly to mathematical multiplication of zero, produces again empty service.

Properties (3.98)-(3.101) are commutativity properties, which we have already declared or proved within the sections devoted to corresponding operators. Only binary operators can be commutative. Commutativity properties indicate arbitrariness of operands order with the corresponding operators. These properties can be useful during composite services' modeling when manipulating complex algebraic expressions and when the order of operands is important for performing certain algebraic manipulations.

Properties (3.102)-(3.106) are associativity properties, which we have already declared or proved within the sections devoted to corresponding operators. Only binary operators can be associative. Associativity properties indicate arbitrariness of operators' execution order for two or more consecutive operators of the same

type. These properties can be useful during composite services' modeling when manipulating complex algebraic expressions and when the order of operators is important for performing certain algebraic manipulations. Associativity properties can be especially useful during hierarchical modeling when the order in which component services are nested into each other with the same type of operator really matters. So, if the corresponding operators are associative the order of services' inclusion can be easily reverted.

Properties (3.107)-(3.117) are distributivity properties. We have not studied these properties previously since distributivity is the property of operator pairs. Let us give then a definition of distributivity.

Definition 3.18. Provided two binary operators * and + are defined on a set S and given elements x, y, and z such that $x, y, z \in S$, it is possible to say that:

- Operator * is *left-distributive* over operator + if

$$x * (y + z) = (x * y) + (x * z) \tag{3.122}$$

- Operator * is *right-distributive* over operator + if

$$(x + y) * z = (x * z) + (y * z) \tag{3.123}$$

- Operator * is *distributive* over operator + if it is both left- and right-distributive over this operator.

It should be noted that if the operator * is commutative, then conditions (3.122) and (3.123) are logically equivalent.

Considering properties (3.107)-(3.117) it can be noted that only two operators in our algebra are such that other operators can be distributed over them. These two are the mutual exclusion and the parallelism operators.

Distributivity properties are, perhaps, the most powerful tool of composition complexity reduction and structure optimization. Application of distributive transformation to service composition formulas can often remove redundant component service entries from the composition structure. However, along with algebraic or structural optimization there also exists the technological aspect which must be considered. We can call it execution optimization. Execution of a composite service cannot always be optimized by applying distributive rules, in some cases this can be even prevented. Formulae (3.115)-(3.116) can be thought of as examples of questionable distributivity in terms of service execution. When transforming the service structure from the one presented on right-hand side of these formulae to the left-hand side, the execution of the composite service might be transformed incorrectly. The reason for that can be, for instance, the existence of certain functional dependencies within the pairs of sequenced services which will be no longer valid after such transformation. Unfortunately, such dependencies are often characterized as implicit semantic relationships and cannot be formalized

within our approach. So, the properties (3.115)-(3.116) should be used with special caution.

We can see that all the operators of our algebra except iteration and parallelism with communication operators can be distributed over the mutual exclusion. Moreover, they can be distributed and collapsed in a correct manner both from a structural and execution viewpoint. However, distributivity over the mutual exclusion operator is beneficial only from the structural viewpoint because it allows removing redundant service entries when collapsing the structure with respect to the distributivity rule. Nevertheless, in terms of execution effectiveness nothing is gained, since the mutual exclusion in any case executes only one alternative out of two, so execution redundancy is not even present.

Let us now show that left-distributivity property (3.111) of the unordered sequence over the mutual exclusion holds.

First, we take the left-hand side of the formula (3.111) and transform it according to the equality (3.118).

$$S_1 \Leftrightarrow (S_2 \otimes S_3) = (S_1 \triangleright (S_2 \otimes S_3)) \otimes ((S_2 \otimes S_3) \triangleright S_1) \tag{3.124}$$

Now we twice apply the distributivity rule (3.107) on the right-hand side of the equation (3.124).

$$S_1 \Leftrightarrow (S_2 \otimes S_3) = ((S_1 \triangleright S_2) \otimes (S_1 \triangleright S_3)) \otimes ((S_2 \triangleright S_1) \otimes (S_3 \triangleright S_1)) \tag{3.125}$$

Open the parentheses on the right-hand side of (3.125):

$$S_1 \Leftrightarrow (S_2 \otimes S_3) = (S_1 \triangleright S_2) \otimes (S_1 \triangleright S_3) \otimes (S_2 \triangleright S_1) \otimes (S_3 \triangleright S_1) \tag{3.126}$$

Recomposing the sequences on the right-hand side of the equation (3.126) as follows:

$$S_1 \Leftrightarrow (S_2 \otimes S_3) = ((S_1 \triangleright S_2) \otimes (S_2 \triangleright S_1)) \otimes ((S_1 \triangleright S_3) \otimes (S_3 \triangleright S_1)) \tag{3.127}$$

and then collapsing the right-hand side of the equation (3.127) according to equality (3.118) we obtain:

$$S_1 \Leftrightarrow (S_2 \otimes S_3) = (S_1 \Leftrightarrow S_2) \otimes (S_1 \Leftrightarrow S_3), \tag{3.128}$$

which was to be proved.

It is not necessary to prove the right-distributivity property (3.112). It holds because we proved that (3.111) holds and the operator of the unordered sequence is commutative as reflected by (3.100).

As regards distributivity over parallelism, the only distributive operation here is the sequence. Nonetheless, this type of distributivity, despite its doubtful execution correctness, can help composite service become more optimized in terms of execution as well as from the structural viewpoint. When collapsing the service structure with respect to rules (3.115) and (3.116) not only redundant service

entries are removed from the service structure but also redundant service entries are removed from the execution process of the composite service.

Although the sequence operator is not commutative, it is both right- and left-distributive over the operators of mutual exclusion and parallelism as reflected by properties (3.107)-(3.108) and (3.115)-(3.116).

An especially interesting property here is the distributivity of the refinement operator over the mutual exclusion. The refinement operator is not commutative and as a consequence it is only left-distributive over the mutual exclusion as reflected by formulae (3.113)-(3.114). In addition to that, the distribution rule (3.113) should be also applied very carefully since refinement with mutual exclusion and the mutual exclusion of two refinements are not always the same, i.e., the difference between selection of the refining service in advance or during the execution of the refined service may make sense in some cases.

The parallelism operator with communication cannot at all be distributed over other operators. Neither other operators can be distributed over it. The reason for that is the presence of the communication set, which is particularly designed to facilitate synchronization over a pair of services, i.e., it can be algebraically correct only when applied over a binary operation. As soon as additional services become subjected to the parallelism with the communication operator it can be no longer verified that the communication set is constant over algebraic transformations. Neither can it be guaranteed that it does not contain newly emerged communication locks.

Property (3.117) distributes the iteration operator over the parallelism operator with communication. This is the only exception from the previous statement. Such distribution is possible due to the fact that the iteration operator is unitary, so it brings no additional service into the scope of the parallelism operator.

Property (3.119) is obtained by consecutive application of properties (3.118) and (3.96).

Property (3.121) refers to the case when the refined service operation is not set or cannot be found within the refined service. Then the refined service remains unchanged.

The last but not the least observation concerning our Web service algebra is that it verifies the closure property, i.e., application of each defined algebraic operator to services produces services, to which the algebraic operators can be again applied. Thus, it is possible to use declarative expressions of the service algebra to define any imaginable composite service in terms of component services and reuse previously built service declarations for building more complex services. This exactly satisfies the established requirement of service definition recursiveness.

So, at first services are defined as algebra declarations. Then they are modified using algebraic properties to optimize the structure of the corresponding services. Only after the completion of all necessary algebraic transformations

service declarations are mapped into Petri net models for further validation and analysis.

3.4 Web Service Analysis

Web services generally represent complex well-structured systems consisting of possibly numerous units. Though these units, component services, are designed for autonomous operation, they are combined and reused within composite service systems they are included into. As we previously observed in this Chapter, the ways of such cooperation can be somewhat different in a logical and functional sense. Assuming that components services themselves are neatly designed lacking logical and functional errors, it is still not a trivial task to design a correct composite Web service from the correctly built component services. Besides functional correlations between component services which may be hard to formalize, and logical dependencies between them which are often difficult to discover, there are also behavioral aspects of Web services' operation that are even more difficult to specify and control, especially taking into account possible examples of non-deterministic behavior when output may vary over the same initial conditions. However, the majority of these aspects are related to rather high-level analysis problems and therefore are beyond this Chapter's scope. This Chapter is more focused on service composition syntax rather than semantics, though semantics is not totally abandoned. We believe that correct syntax is the absolute prerequisite for building correct semantics.

Petri nets are an especially useful technique for modeling of systems that exhibit concurrency. Composite services, as we define them, are belong to these types of systems. Therefore, we utilize Petri nets for their modeling. Modeling by itself is, nevertheless, of little use, if it is not accompanied with an appropriate analysis. Analysis is what evaluates and validates modeling efforts making them beneficial against direct systems' implementation. Since we model services using Petri nets, the analysis should also be done by means of Petri nets analysis techniques.

The correctness of Petri net models is determined by several important properties of Petri nets, which we formally discussed in the previous Chapter. In the context of service, Petri nets being the subject of analysis, some of the discussed properties are more important to verify, some are less. Below we discuss how the commonly analyzed properties of Petri net models map into execution properties of the corresponding services. We will not give any formal definitions of the properties here, since they were already given in Chapter 2. Instead, we will concentrate on service-related aspects of corresponding properties.

3.4.1 Safeness and Boundedness

As previously discussed, safeness and boundedness are conceptually the properties with the same meaning.

Definition 3.19. A place of a Petri net is *safe* if the number of tokens in it never exceeds one for any reachable marking of the Petri net. Accordingly, the Petri net is safe when all its places are safe.

Definition 3.20. A place of a Petri net is *k-safe* or *k*-bounded if the number of tokens in it never exceeds $k \geq 1$. The Petri net is *k*-safe if all its places are *k*-safe. It is important to note that the *k*-safe place is also *n*-safe if $1 \leq k \leq n$. Then the bound *k* for the Petri net can be defined as the maximum bound over all its places.

So, safeness is basically a special case of boundedness. However, safeness property is quite important for our definition of Web Services, because in the majority of cases they are defined such that places in service nets have to be safe. Firstly, we use ordinary Petri nets for service modeling, which do not allow multiple arcs between places and transitions. Secondly, within most of our operators there are only few cases where a place may have more than one incoming arc, i.e., can potentially violate the safeness property. But even in those cases composition constructs are defined so that they are carefully using pairs of conflict and concurrency *fork* (split) and *join* (merge) constructs (see Figure 66) in order to bifurcate process execution in such way that it allows for encapsulation of a single token within each execution branch, thus, preserving safeness.

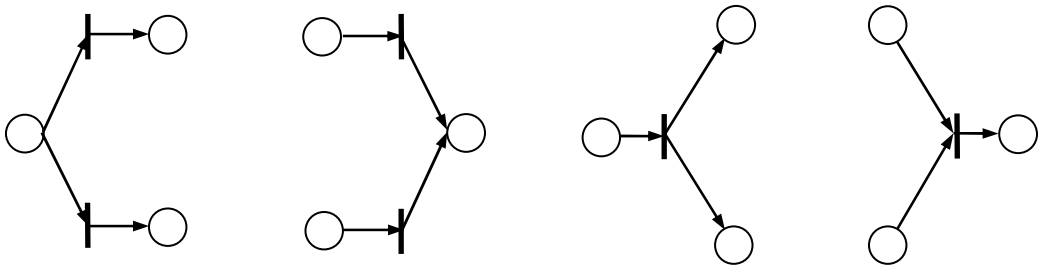

FIGURE 66 Fork and Join constructs of the type a) conflict and b) concurrency

The only construct which unavoidably establishes an unsafe place is iteration. It includes the special control place which regulates the number of iterations. Although this place is generally not safe, it is still bounded according to the constructs' design, i.e., the number of iterations performed by the iteration operator is finite. If it were unbounded, then the number of iterations would be potentially infinite resulting in a potential livelock in the service net. Fortunately, infinity cannot be formalized using Petri nets. Another interesting observation is that when we remove the iteration control or move it outside the service net thus

switching from a restricted iteration operator to an unrestricted one, the safeness property holds for all the places belonging to the corresponding construct.

As we already mentioned our service algebra is extensible in the sense that new operators can be introduced into it if they allow building new Web Services on the base of other Web Services, i.e., can be included into our algebra's closure. An example of a useful operator that can be added to the service algebra is the *competition* operator. This operator composes two services executed concurrently, but completion of either of them is sufficient for completion of the composite service. The corresponding service net is shown in Figure 67. The reason why we introduced such operator is that there is a place in the corresponding service net which is potentially unsafe. Its safeness is violated due to the fact that the elements of the used Fork-Join pair are of different type. So, the corresponding place, marked with *a* in Figure 67 may contain two tokens if it happens that both component services are completed simultaneously. Still this place is bounded by the number of component services composed by the corresponding competition operator.

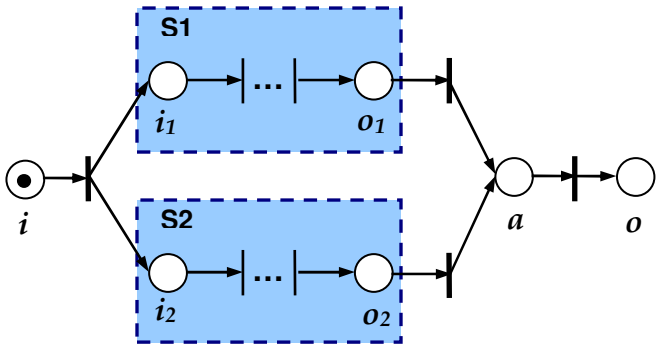

FIGURE 67 Competition of two services

Here we can conclude that although service nets are not always safe they are always bounded. Therefore, safeness and boundedness are important service net properties, which should be checked in order to detect possible errors committed during structural design of services. In an ideal case any service net must be safe, in other cases at least bounded.

3.4.2 Conservation

Conservation property can be especially important for modeling of resources and resource allocation systems. In case of such systems, services may hold and release certain computational or material, possibly shared, resources. Tokens in the corresponding Petri net models can represent such resources. It is sometimes important that tokens are neither destroyed nor created during the execution of services, since resources cannot be destroyed or created by the service using them

either. This highlights the importance of the conservation property. We give a somewhat simplified definition of conservation.

Definition 3.21. A Petri net is *conservative* if the total number of tokens present in the net is constant in any reachable marking.

According to this definition, the conservation property is often violated when composing services using our service algebra. At least the parallelism operators duplicate tokens with their input forks. However, they also successfully absorb duplicated tokens with their output joins.

A good example of a case where conservation property is important would be concurrent resource allocation. Let us introduce a special construct named *parallelism with sharing*, which is especially devoted to modeling of concurrent resource allocation. Such construct implies that two services are executed concurrently, but their execution is synchronized by the availability of certain resource. Both services intend to use that resource, but the resource cannot be allocated to both services simultaneously, i.e., one service must wait until the other service releases the resource. An example of a use case can be a blackboard based communication of two people, who have a single piece of chalk to perform a dialogue using a blackboard. The corresponding construct works similarly to the parallelism with communication construct with the difference that while parallelism with communication models direct mutual dependence, parallelism with sharing models mediated mutual dependence, i.e. two services are mutually dependent through a mediation point, e.g., a shared resource. The corresponding service net of the parallelism with sharing construct is shown in Figure 68.

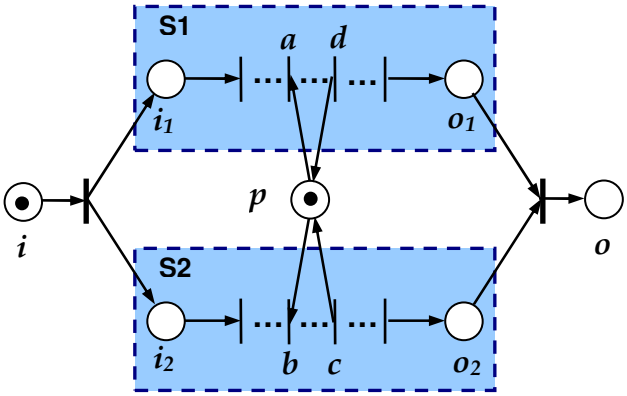

FIGURE 68 Non-conservative parallelism with sharing of two services

However, it can be seen that the parallelism with sharing construct in the way it is designed in Figure 68 is not conservative. The token residing in the place p, which indicates resource availability, is absorbed whenever one of the services captures the resource. The token representing the resource is reproduced into the place p as soon as the service releases the resource. So, although the resource allocation is in

principle modeled correctly, the corresponding service net violates conservation property. This consequently may result in the impossibility of resource allocation correctness validation. Therefore, the parallelism with sharing construct, if it is to be introduced into the service algebra, should be modified to preserve the conservation property. There are multiple ways of making a conservative version of this construct. For example, all the arcs within the execution sequence of each service between the resource capturing operation and resource releasing operation can be doubled. This would lead to the preservation of conservation property. However, this method would also result in a violation of the Petri net type limitation. We declared that we use ordinary Petri nets for service modeling, but such arc duplication is prohibited by this type of Petri nets. So, we prefer another way of modification. We introduce one more auxiliary place to our parallelism with sharing construct. The role of this place is to indicate that the resource is busy, while the role of the old place p is to indicate that the resource is available. Figure 69 illustrates the conservative version of the parallelism with sharing construct.

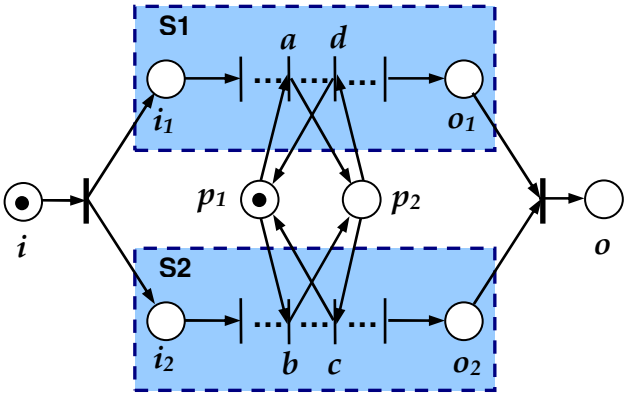

FIGURE 69 Conservative parallelism with sharing of two services

Indeed, now the token representing the resource resides either in the place p_1 indicating that the resource is idle, or in the place p_2 indicating that the resource is busy. So, the corresponding service net always has precisely two tokens and hence is conservative.

We conclude that according to our modeling methodology the conservation property as it is defined above is of little use. It will be violated very frequently as concurrency is present within services built. However, it can be useful to check the conservation property of certain subnets within composite service nets, where conservation is vitally important. The subnets modeling concurrent resource allocation are good subjects for testing of conservation property. Such analysis discovers if the corresponding services will be correctly manipulating real resources.

3.4.3 Liveness

Liveness is, perhaps, the most important property of service nets to be analyzed. If a service net is not live the service modeled by it is simply incomplete, i.e., it cannot produce desired results or even be correctly terminated.

If a Petri net is not live, it is dead. This means that the Petri net contains a deadlock, the situation where no transition belonging to the net can be fired. In terms of Petri net formalism, a deadlock is a marking in which there are no enabled transitions.

So, in order for a service to be accomplishable its service net must not contain deadlocks. However, this is not exactly a strict condition because due to a possible non-determinism a deadlock may affect only certain intermediate transitions turning them dead, while the whole composite service can be still accomplished. This situation can be called a *potential* deadlock.

Formally, liveness is a non-trivial property. As we observed in the previous chapter, liveness can be characterized with levels. Below we describe these levels informally, since we already gave their formal descriptions in Chapter 2. Here we characterize not the liveness levels of an arbitrary transition but of the transition preceding the output place of the composite service, assuming that if this transition can be fired, then the service is accomplishable.

Definition 3.22. A transition t_o is *live* at the following levels:

Level 0: the transition t_o cannot be fired, since there is no firing sequence in any reachable marking such that this transition can be enabled. This level corresponds to a deadlock and to a dead service.

Level 1: the transition t_o is potentially fireable, i.e. there is a reachable marking in which this transition is enabled. This level corresponds to a potential deadlock and a potentially live service.

Level 2: the transition t_o can occur within firing sequence a finite number of times. This level corresponds to a finite live cycle.

Level 3: the transition t_o can occur within firing sequence infinitely often. This level corresponds to an infinite live cycle.

Level 4: the transition t_o is always fireable, i.e. for any reachable marking except for the final marking there exists a firing sequence such that this transition is enabled. This level corresponds to a strictly live service.

Now we can say that the necessary condition of the service liveness, i.e. feasibility, is that the transition t_o is live at level 1. The sufficient condition of the service liveness is that the transition t_o is live at level 4.

In the Petri nets theory the level of liveness of the whole Petri net is usually defined as the minimum liveness level over all the transitions of the Petri net. Our service nets follow a sort of specific topological design, which dictates that the transitions within service nets are not all equitable in the sense of their importance.

The reason for that is first of all a possible non-determinism expressed by a service net. So, service net liveness is not necessarily a linear function of its transitions' liveness levels. The service net liveness level is generally equal to the liveness level of the transition t_o.

The transition t_o can be live at level 1 even if some other transitions in a service net are dead. Levels 2 and 3 are basically irrelevant to the case of the transition t_o because this transition cannot be a part of finite or infinite cycles. However, if other transitions are live at levels 2 or 3 they can still lead to a deadlock in case of a finite cycle or to a livelock in case of an infinite cycle. The transition t_o is live at level 4 if all the transitions in a service net are live at least at level 1. Liveness at level 4 for the transition t_o means that this transition is enabled in the marking, which is always reachable from any other reachable marking, or in the set of markings, one of which is always reachable from any other reachable marking. The exception for this rule is the final marking. The final marking is a marking in which the output place o of a service net contains a token. Due to the principle of service modeling we established, no other marking can be reached from the final marking, i.e., no transitions are enabled in this marking. So, the final marking is basically a deadlock, but a legally permissible deadlock.

Our principles of service net design, e.g., Petri net formalizations of the composition operators, do not generally allow for deadlocks. Deadlocks usually arise when corresponding Petri nets contain cycles and allow for multiple arcs, which is not the case for service nets. The only legitimate type of cycle is established by the iteration operator. However, it is built in the way which excludes the possibility of deadlock emergence. Deadlocks may appear if some of the component services are incorrectly designed or when using complex composition operators such as the parallelism with communication, where the possibility of a communication lock remains.

3.4.4 Reachability

Reachability problem is also a very important problem of service net analysis. In general, reachability problem sets the following question: is certain marking reachable from another marking? In the context of service composition we are interested in answering the particular question: is the final marking reachable from the initial one?

It can be easily seen that the reachability problem is closely related to the liveness problem. There is a strong relationship between them. A service net is live at least at level 1 if the final marking is reachable from the initial one, and vice versa.

In order to solve the described Petri net problems certain analysis techniques are needed. Moreover, such techniques should allow computer-aided automatic analysis of Petri net models.

Two major techniques for analysis of Petri nets have been suggested. The more commonly used analysis technique is the *reachability tree*; while the other one is referred to as *matrix equations*.

We believe that the use of the reachability tree technique is absolutely sufficient for the purposes of service composition models analysis, since it can be applied for successful solution of all the above described analysis problems. While safeness, boundedness and conservation problems are easily solved by the reachability tree technique, the technique can encounter difficulties while solving liveness and reachability problems. However, should such difficulties arise, they are always explained by the presence of infinite live cycles in the analyzed Petri net necessarily resulting in infinite token duplication. Reachability tree has no power to adequately cope with token duplications by distinguishing the number of duplicated tokens. Instead, it substitutes their number with a special variable ω meaning infinitely large number of tokens. If a duplication cycle leads to a deadlock, this situation cannot be correctly modeled by the reachability tree. So, liveness property cannot be properly checked in such cases. The same problem may occur when checking reachability of a certain marking in the presence of infinite duplication cycles, since the reachability tree cannot make difference between the duplication sequence incrementing the number of tokens by one and the sequence incrementing this number by any other amount.

Nevertheless, we define service nets in the way which does not allow presence of duplication cycles, so liveness and reachability problems must be successfully solvable with the aid of the reachability tree technique.

Figure 70 shows a sample service net, and Figure 71 presents the reachability tree for this service net.

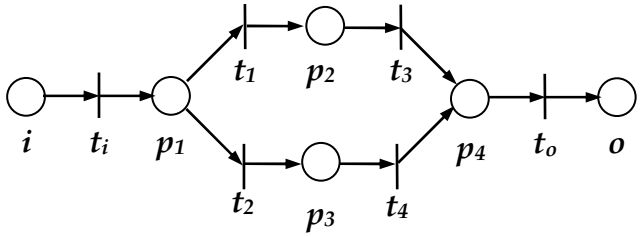

FIGURE 70 Sample service net

As Figure 70 illustrates a simple but a typical example of a service net, the reachability tree shown in Figure 71 allows verification of this service net's properties. It shows that the service net is safe, conservative and live at level 4. Safeness is indicated by the fact that in the reachability tree there are no markings with elements exceeding one. Conservation is proven by the fact that the sum of the elements of each marking is constant and equal exactly one over all the reachable markings. Finally, liveness at level 4 is verified by the fact that all leaves of the reachability tree are identical and represent the final marking of the service

net. If the reachability tree contained also leaves with markings different from the final, it would mean that there is a deadlock within the service net and the service net is live at level 1. If the reachability tree contained only leaves with markings different from the final one, it would mean that the service net is live at level 0, i.e. is dead. Reachability of any possible marking from any reachable marking can be tested by checking two things: the presence of the corresponding marking in the tree, and the existence of a path or a set of paths to this marking from the marking against which reachability is tested.

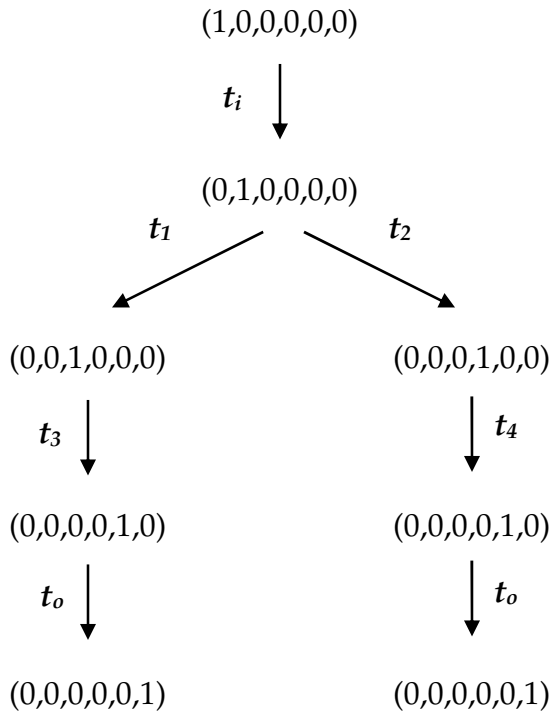

FIGURE 71 The reachability tree of the service net of Figure 70

3.5 Concluding Remarks

In this Chapter we utilized the Petri nets formalism as a formal tool for modeling Web services and service composition. We gave a recursive service definition allowing building composite services out of other composite services. Then we introduced a minimal set of composition operators to be applied for composite service design and gave them appropriate formalizations in terms of Petri nets. We also proposed several advanced composition operators that increase the modeling power of the basic service algebra and easily map into Petri net representation too.

We justified the choice of advanced composition operators by presenting appropriate use cases and application examples. Then we examined the methods of service composition and showed examples of composite services' building using various composition operators. We studied algebraic properties of the suggested operators and described algebraic mechanisms of structural and execution service nets optimization. Finally, we investigated the issue of how composite service nets must be analyzed and validated to ensure proper operation and reliability of implemented composite services.

Petri nets appear to be quite a suitable tool for modeling services as execution processes. Its major advantages are pictorial clarity, possibility of formal analysis and, most importantly, facilitation of concurrency and non-determinism modeling. There are some scalability concerns which may arise when speaking about huge heavily populated compositions. However, this drawback can be partially compensated by performing optimization on the level of proposed service algebra.

The suggested service algebra features clear and concise notation, which alleviates possible scalability concerns. Moreover, it is easily mapped into Petri net representation, which is convenient for performing necessary structural optimization efforts on the level of algebraic constructs. Additionally, the service algebra verifies closure property on the set of proposed operators, which allows hierarchical service composition, and is easily extensible with additional types of operators, which can significantly increase its modeling power.

As the result, we developed a complex formal solution technique for the service composition problem. Our modeling framework allows hierarchical construction and construction from scratch of structurally and execution-wise efficient, error-free, and complex composite services which express concurrency and non-determinism.

As the base for that we used the work of Hamadi and Benatallah [49], which we extended noticeably in many aspects. More specifically, we modified the set of composition constructs and the particular layout of some of the constructs presented in [49], we rigorously studied the use cases for each composition construct, we thoroughly investigated formal properties of the composition operators and showed how they can be used to optimize composite service structures, finally, we described the means for formal analysis and verification of Petri net based service models.

4 CONTEXT UTILIZATION FOR WEB SERVICE COMPOSITION

In this chapter we present the main contribution of the thesis, the Context-Aware Web Service Composition framework. We will describe the entire process of Web Service composition and indicate how we envision the usage of context for some of its critical procedures. We will present the concept of (Composite) Context-Aware Web Service and model it, as well as the corresponding composition procedures, using the constructs and operators described in Chapter 3.

4.1 Context-Aware Web Service Definition

In Chapter 3 we define Web Service as a functional unit transforming input data into output data (see Figure 24). We said that each service can be represented as a process of sequential state changes and is associated with a service function which transforms service's initial state into its final state. The service function can be arbitrarily complex in the sense that it can be decomposed onto more primitive service functions and may correspond to real service operation(s).

Here we will try to extend the Web service definition given in Chapter 3 to cover the class of Context-aware Web Services.

As we concluded in Section 2.3 a context-aware service is a service which is capable of changing the structure of its execution flow with respect to context.

Then the change of service behavior corresponds to the change of the respective set of service operations. It is obvious that in such a definition any context-aware Web service has to allow at least two different sets of service operations which are mutually exclusive and selected with respect to current context. Remembering that we used to encode sets of service operations as service functions we can now define a context-aware Web Service as follows.

Definition 4.1. *Context-aware Web Service* is a complex functional unit having an implicit structure and explicit inputs and outputs that is capable of performing two or more mutually exclusive service functions $F_1,\ldots,F_n$, which transform its input parameters X into its output parameters Y and are selected by the context function $f=CCF(F,C)$, where:

- f is the selected function out of $F_1,\ldots,F_n$;
- CCF is the context control function;
- F is the vector $(F_1,\ldots,F_n)$;
- C is the vector of contexts $(c_1,\ldots,c_m)$.

The graphical representation of the corresponding construct is shown in Figure 72. In this figure the thick red arrow encodes a control influence, to which a mutual exclusion operator (depicted with a crossed circle) is subjected.

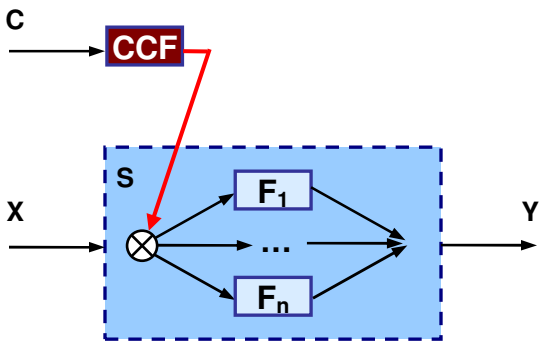

FIGURE 72 Context-aware Web service

An important note concerning the given definition is that it defines context-aware Web service so that the service uses context merely as an external control facility and not as input data. However, in the previous section it was mentioned that context-aware service can generally use context both as control and as data. Our definition of context-aware Web service does not, nonetheless, contradict such an understanding of context awareness. This is simply explained by the fact that our designed Context-aware Web Service Composition framework does not model data flows between Web services, but models merely control flows between them. The modeling of data-oriented context usage is beyond the scope of this thesis, but is still possible.

Another interesting observation and the result of the given definition is that, according to the Web service definition given in Chapter 3 context-aware Web service cannot be an atomic service, since it necessarily consists of more than one service function. Thus, a context-aware Web service can be either simple or composite.

A simple context-aware Web service (see Figure 72) comprises multiple distinct service functions hardcoded into the service. At least two of these service functions have to be alternative to one another in order for the service to become context-aware. However, this type of context-aware Web service does not interest us here, because, firstly, it has no connection with service composition, and, secondly, it cannot be created automatically but only by manual human design. Moreover, an automatic matchmaking procedure that selects between the alternatives based on the context cannot operate with service functions since they are transparent for outside view of the service. Such a procedure can only operate with services as chunks of service compositions.

A composite context-aware Web service is what we will further concentrate onto. Recalling our classification of composite Web services from Chapter 3 into purely composite and partially composite, we now limit our interest to purely composite services, and omit the discussion of partially composite context-aware services for the time being. The reason for that is in possible transparency of some contextually controlled alternatives which may correspond to the service's own functions just as in the case of simple services.

In case of a purely composite service all possible execution alternatives are services. These services can be subjected to a contextual control, and this control can be performed in an automatic fashion by a composition reasoning procedure matching acquired context with contextual requirements of component services, as we show in the subsequent sections. Figure 73 illustrates the case of composite context-aware Web service.

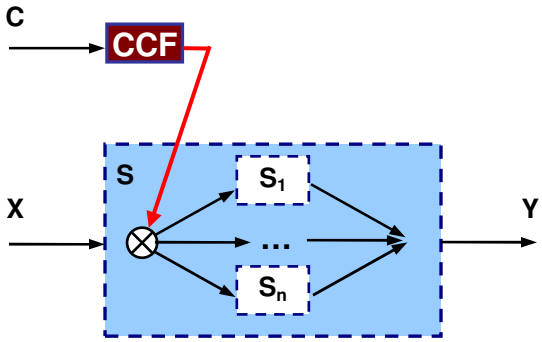

FIGURE 73 Composite context-aware Web service

In case of composite context-aware service the context control function schematically can be written as follows:

$$s = CCF(S_1,...,S_n,c_1,...,c_m) \tag{4.1}$$

where:

- S_i are alternative component services;
- c_j are the contexts influencing the choice between the alternative services.

In both cases described in Figures 72 and 73 the context control is external to the service, which might not be always acceptable. If we aim at fully autonomous context-aware Web services, then the contextual control has to be incorporated into the actual service. It is not difficult to achieve if the service control structure is supplied to the service at design time. By service control structure we specifically mean a set of context control functions and control links between them and the corresponding mutual exclusion splits. The corresponding structure of the autonomous context-aware Web service is shown in Figure 74. In this figure the thick red arrow encodes a control link, the ordinary arrow labeled with *C* encodes context data flow, and the dashed red arrow labeled with *R* models the data flow of contextual requirements posed by the component services $S_1,\ldots,S_n$.

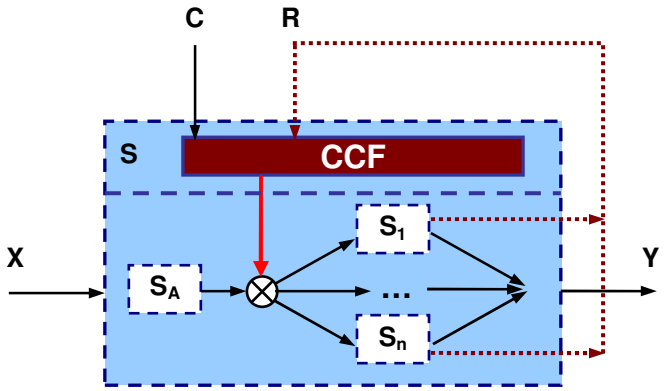

FIGURE 74 Autonomous context-aware Web service

The context control function can be now defined as follows:

$$s = CCF(R,C) \tag{4.2}$$

where:

- $s \in \{S_1,...,S_n\}$;
- R is the set of service requirements:

$$R = \{(r_1^1,...,r_{n1}^1),...,(r_1^n,...,r_{nn}^n)\} \tag{4.3}$$

i.e.:

$$R = \bigcup_i^N \bigcup_j^{N_i} r_j^i \tag{4.4}$$

where each r_j^i corresponds to S_i;

- C is the set of relevant contexts:

$$C = \bigcup_{k}^{M} c_k \tag{4.5}$$

such that:

$$M = |C| = |R| = \sum_{i}^{N} N_i \tag{4.6}$$

where N is the number of alternative services and N_i is the number of contextual restrictions for each service S_i.

So, basically the context control function checks as many contexts as is the number of contextual restrictions established by all alternative services. The job of the CCF function then is to compare current contexts with the corresponding restrictions and select, among the alternatives, the service which exhibits the best match. We will return to this point with the detailed description of CCF functions in Section 4.4.

Figure 74 shows the service with a single contextually controlled structural choice and with a single context control function. However, it is quite probable that in a sophisticated composite service it will be necessary to perform several structural choices based on context. In such a case every single structural choice will be managed by a separate context control function. An example of such a service is presented in Figure 75, where two sets of alternative services are to be executed concurrently.

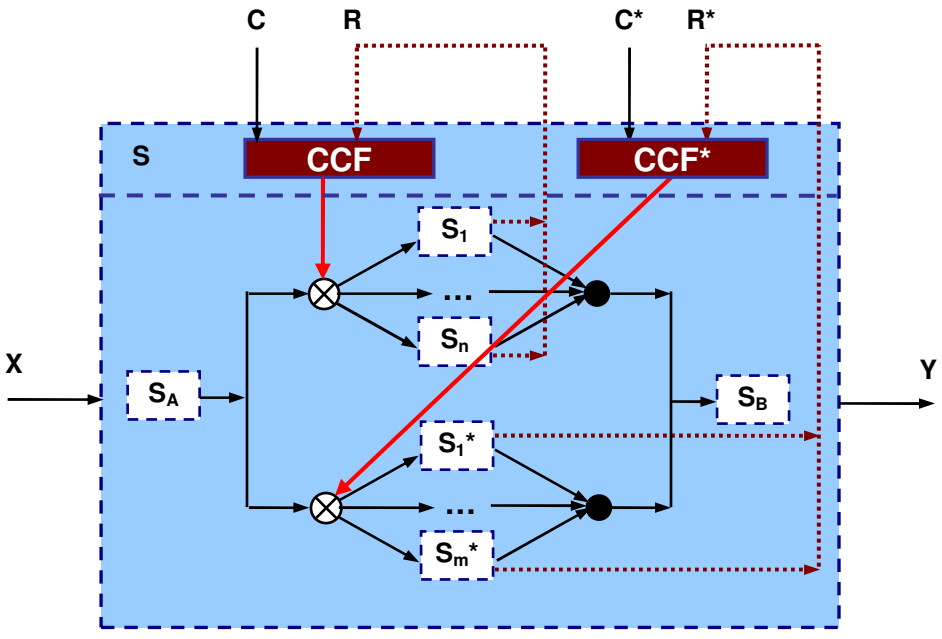

FIGURE 75 Composite context-aware Web service with two context control functions

As can be seen in Figure 75, when two independent sets of alternative component services exist within the composite service structure, two independent choices have to be made. These two choices are performed by two distinct context control functions on the basis of two distinct sets of contexts. In addition to structural independence of choices, they are also independent in terms of execution: they can be performed at arbitrary moments of time, in arbitrary order or in parallel, and in many cases even disregarding the causal order of operations within the managed control flow.

Thus we have showed the general idea of how a context-aware Web service is built and how it operates in comparison with a normal Web service. However, we still should say some words about how a context-aware Web service can be disposed of. Context-aware Web services are like ordinary Web services in terms of invocation. Thus, they can be normally used as component services when composing a complex service and subjected to all existing service algebra operators without limitations.

Figure 76 illustrates a composite context-aware Web service that comprises component context-aware Web services.

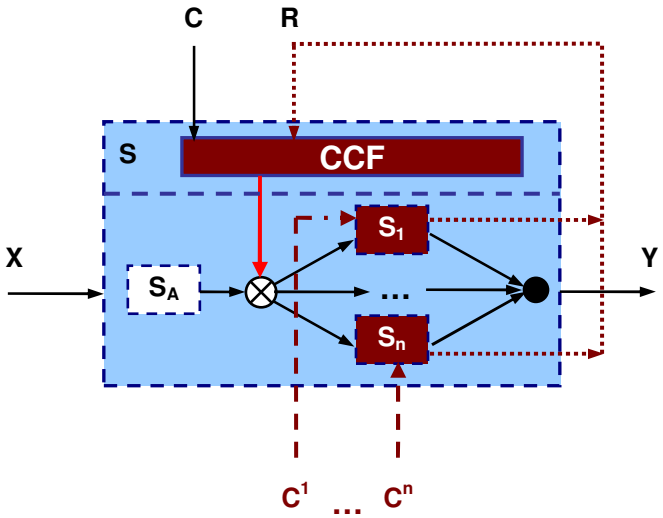

FIGURE 76 Composite context-aware Web service with component context-aware Web services

So far we can see that the main explicit distinction of context-aware Web services is that they have extra context inputs, through which they are able to acquire and utilize necessary contexts in real-time. However, within a composite service structure context, inputs of component context-aware services are transparent for the composite service and therefore are not managed by the composite service's context control facility.

Implicitly, context-aware Web service structure can be distributed over two distinct functional layers. The lower layer called "service layer" constitutes a

normal Web service by specifying its components and control flow linkage between the components. The higher layer called "control layer" implements a contextual control functionality over the exclusion operators on the service layer.

In terms of Web service algebra, which was presented in Chapter 3, all services, whether they are context-aware or not, can be subjected to all algebraic operators. All the algebraic operators manipulate context-aware services in the same way that they operate with ordinary Web services. However, when composing a context-aware Web service the operator of mutual exclusion has a special role. It performs context-based selection among alternative component services. Thus, two modifications of the mutual exclusion operator should be distinguished: basic mutual exclusion $\otimes$ (see section 3.3.1.6) and contextually controlled mutual exclusion $\overset{C}{\otimes}$. Thus, the example of the composite service shown in Figure 76 can be algebraically written as follows:

$$S = S_A \triangleright \left(S_1 \overset{C}{\otimes} S_2 \overset{C}{\otimes} \ldots \overset{C}{\otimes} S_n \right) \tag{4.7}$$

where C is the set of contexts over which the selection is made.

In terms of Petri net representation a service net of a composite service is modeled identically regardless of the presence or absence of the component context-aware services. Figure 77 depicts the service net for a modified example of the context-aware Web service illustrated in Figure 76.

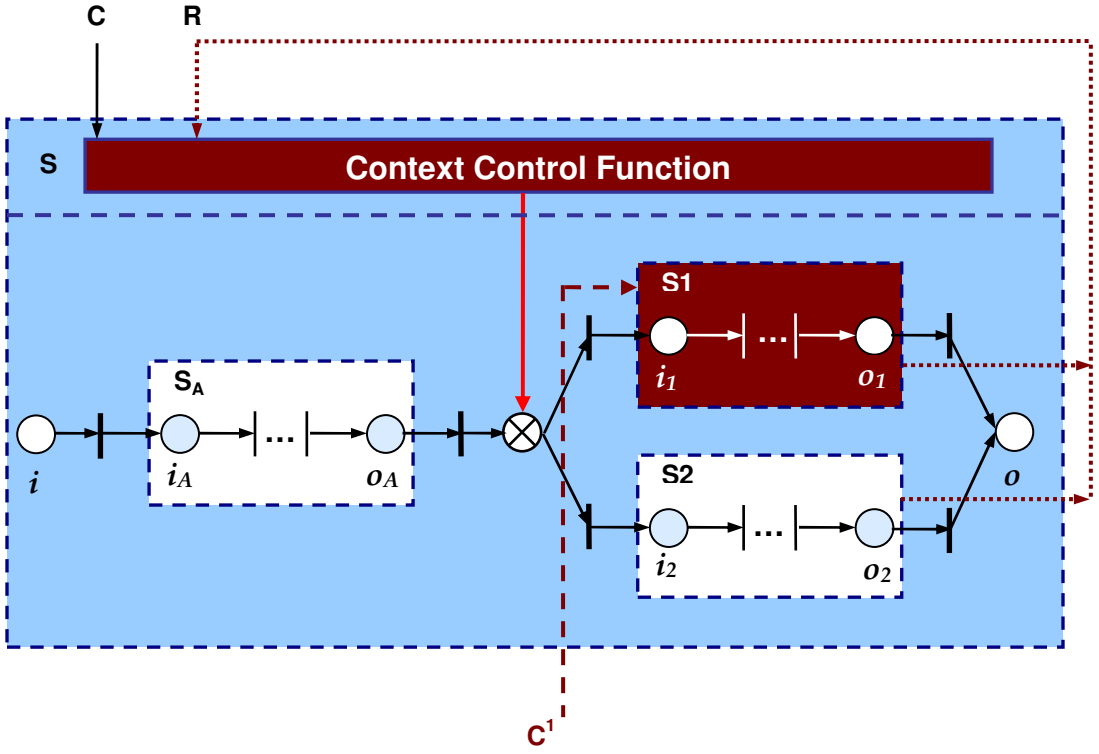

FIGURE 77 Service net of a composite context-aware Web service

In addition to showing that the principle of service net construction for ordinary Web services is well preserved in the case of context-aware Web services, Figure 77 also indicates that ordinary and context-aware services have equal rights and treatment when used as alternative component services within a context-aware composite service. Service S_1 is context-aware, while service S_2 is not. Still they are easily combined by the mutual exclusion operator and identically provide their sets of requirements to the context control function of the composite service S.

In this example and in this entire section we do not show the construction of a *control net*. Using the concept of control net we encode a Petri net-based model of a control layer within a composite context-aware service, and particularly a Petri net model of a context control function. These issues are to be modeled and investigated in depth in Section 4.4 of this Chapter.

Summarizing this section we have to emphasize some important aspects resulting from the definition of context-aware Web service we gave. Normal Web services basically function in a deterministic way, i.e., one set of inputs in principal always produces the same output. Exceptions to this statement may occur if either some degree of randomness is imparted to a service by including operators of randomly controlled mutual exclusion into the service structure, or when failures occur during the service execution. We will further call mutual exclusion "uncontrolled" if the choice between alternatives is made randomly. A context-aware Web service in the given definition is formally non-deterministic. It relies on the use of contextually controlled mutual exclusion operators, which generally produce different execution paths at each run of the service. Alternative execution paths are chosen not randomly, but based on the context, which is acquired at run-time and makes the result unpredictable prior to the service execution. Nevertheless, the choice in this case is grounded in contrast to random selection performed by ordinary mutual exclusions.

Another interesting observation is that context-aware Web services in our definition are, more specifically, reconfigurable services, i.e. services capable of revising the structure of their control flow at the stage of execution. Though reconfigurability usually implies that some components can be added to the structure as well as removed from it, context-aware Web Services in the given definition can only remove alternative execution paths from their control flow structures. The exact procedure of such removal will be described in Section 4.3. Although this reconfiguration is somewhat restricted, we nevertheless will call this type of structural adjustment a *reconfiguration*. Being in general a highly dynamic matter, context facilitates such dynamic reconfiguration of services at run-time.

So, we conclude that a context-aware Web service is a Web service that is capable of run-time utilization of context information to dynamically reconfigure its flow structure through context-based selection of alternative execution paths present within its control flow. More specifically, a composite context-aware Web

service is a context-aware Web service that reconfigures itself by choosing alternative component services with regard to context during its execution.

In forthcoming sections we will concentrate more deeply on the development of Context-aware Web Service Composition framework and discuss the tasks of context-aware Web service composition planning and reconfiguration of context-aware Web service compositions.

4.2 Context-Aware Web Service Composition Framework

Definition 4.2. *Context-aware Web Service Composition* is the process of composing a Web service from other Web services on the basis of actual and relevant context information.

An important reservation has to be made concerning the above definition. Below we shall argue that the result of Context-aware Web Service Composition framework's operation is not necessarily a context-aware Web service as it was defined in the previous section. We will show that an ordinary composite Web service can be produced as well by the designed framework based on context in the most general case.

4.2.1 Requirements and Limitations

In addition to the main goal of the Context-aware Web Service Composition framework, which consists in robust construction of composite Web services from other Web services, we place a set of requirements for the principles of the framework's operation.

The requirements to the designed framework are:

- *On-demand composition*: composition has to be made in response to the user request and whenever it is necessary.
- *Goal-driven composition*: composition has to be built in accordance with the highly specific goal established by a user or an agent acting on the user's behalf.
- *Context-aware composition*: composition has to be made regarding the current context of the user, of the established goal, of the component services and possibly of the environment.
- *Verifiable composition*: a composite service has to be analyzed and verified before starting the actual execution in order to detect possible errors made during control flow construction.

- *Reconfigurable composition*: the structure of a composite service's control flow has to be capable of undergoing reconfiguration at the time of execution without rebuilding it from scratch.
- *Automated composition*: composition has to be performed mainly automatically with only little interaction with a human user (whose role is ideally reduced to mere specification of composition requests). We specifically target automatic control flow construction.

The above stated constructive requirements shape the designed framework by indicating what we should do and which results we intend to achieve. Still we have to describe some important limitations, which we apply for the design of the composition framework, to explicitly claim what we do not intend to do in this work. The major limitations of the proposed formal framework are the following:

- *Limited data modeling*: we do not explicitly model most of various data to be used in the process of Web Service composition. More specifically, we do not model context, services' inputs, outputs, preconditions and effects. We only model some of the procedures utilizing these data to produce composite Web Services, while particular data models remain implicit for our framework. We however present a formal method for modeling of composition goals. We also pose a requirement that all service-related data, e.g., services' capability and interface descriptions have to be semantically described using Semantic Web languages because it is absolutely necessary to build automatic control flow construction procedure based on modeled composition goals.
- *Limited data flow modeling*: data flow is something we do not model in a comprehensive way, though we neither completely omit this aspect of service composition. This limitation is partially explained by the previous data modeling limitation. We model data flow among component services inside a composite service to the extent that is necessary for construction of adequate and consistent control flow of the composite service. In the majority of cases data flow coincides with the corresponding control flow, and control influences are transferred between services along with data using message exchange. For these types of scenarios it is possible to say that we implicitly model data flows. However, sometimes there may appear specific situations where data are transferred between services using different scheme from that of the control influence propagation. We do not specifically model these sophisticated cases, and neither do we provide specialized modeling tools for doing that. Therefore, most of the above presented requirements to the framework primarily concern control flow construction process, while the requirements to data flow modeling

(similar to those presented in [74]) are simply not presented since such modeling is not dealt with here.

- *Limited technology binding*: when describing the framework we do not provide strong linkage of its components to any certain technological basis. We do not require any specific languages for semantic service description, goal description, Web Service orchestration and for even semantic reasoning to be utilized for the implementation of this framework. We only recommend some of the existing languages as examples of possible technological basis for such implementation. This is due to several reasons. First of all, in this way we guarantee that our solution is universal enough to be applicable to a variety of SOA (see Section 2.1) while requiring only minor adjustments in order to be implemented on top of specific architectures. Secondly, we initially strived for a completely formal framework independent of the specificity of various Web and Semantic Web languages and standards, which is a plus considering the dynamism of these standards' evolution. Finally, in the current version of the framework we do not provide any specification of a semantic composition reasoner, which should be seen as the core part of the possible framework's implementation. This reasoner is supposed to be based on specific technological standards and languages performing the tasks of semantic inference and translation between different languages.

4.2.2 Context-aware Composition Planning

Context-aware Web Service Composition framework being developed in this thesis has basically two operation stages just the same way as most other service composition frameworks. From the composition viewpoint these stages can be called design stage and execution stage, while from the service-oriented perspective more appropriate names would be service creation and service delivery. However, we call these stages composition planning and composition reconfiguration (in this particular case we use the term "composition" not in the sense of the process as we use it for "composition planning", but in the sense of the result produced by this process) respectively with respect to manipulations compositions undergo on these stages.

The planning stage starts on receiving a request for composition and ends up with the composite service ready for execution. In this subsection we briefly describe the general principle of Context-aware Web Service Composition framework's operation on the stage of composition planning. Then we continue with the detailed description of the composition planning stage in Section 4.3.

The process of service composition planning is more or less the same for all known full-automated service composition approaches [58]. It usually comprises

four major phases of preparing compositions: composition request processing, service discovery, service matchmaking, and actual control flow construction. The common practice is the following. At the request processing phase the composition reasoner, which is usually the core part of composition frameworks, receives a composition request, extracts the goal of composition from it, processes the goal and finally obtains a certain conceptual view of the given task. Then in the next phase it discovers appropriate services that potentially fit into certain places within the produced goal-oriented plan. As soon as the services are discovered the reasoner proceeds to the matchmaking phase where it filters away inappropriate services based on their semantic service descriptions. Finally, the reasoner constructs the control flow model of the requested composition and fills it with concrete service instances.

Though we generally follow the described operation principle when designing the process of context-aware composition planning, it has some essential distinctions from the generic planning approach. We illustrate the special features of our planning framework with the aid of the flowchart presented in Figure 78.

The main peculiarity of the planning process presented in Figure 78 is that it includes some additional context-enabled procedures, which are missing in the generic composition planning framework. More specifically, our approach adds one more context-aware service matchmaking procedure, which we call “service delegation”, and an extra operation cycle. In contrast to the generic planning process, our context-aware planning framework generally operates in two consecutive operation cycles: the first is devoted to the service layer functionality construction, and the second builds up the control layer functionality, i.e., the context control function, on top of the service layer built in the first cycle. In other words, the first operation cycle creates an ordinary composite Web service, while the second one makes this service context-aware. In the most general case the second operation cycle performs the same set of procedures to produce the composition of contextual control services on the control layer if such composition is required. The second operation cycle is, however, not always necessary. It is fully engaged only in cases when the service layer contains such alternative component services that the preference can be given to none of them prior to the actual execution. If the number of such alternative service sets on the service layer exceeds one, the second operation cycle may repeat part of the planning procedures multiple times to construct a separate context control function for each set of alternative services.

To illustrate the general principle of the planning process operation and to clarify the above mentioned context related issues, we now describe the process of context-aware service composition planning in a step-by-step fashion. The corresponding steps are indicated in Figure 78 as circled ordinal numbers. The circles corresponding to the first cycle are in light blue, while the circles corresponding to the second cycle are in dark pink.

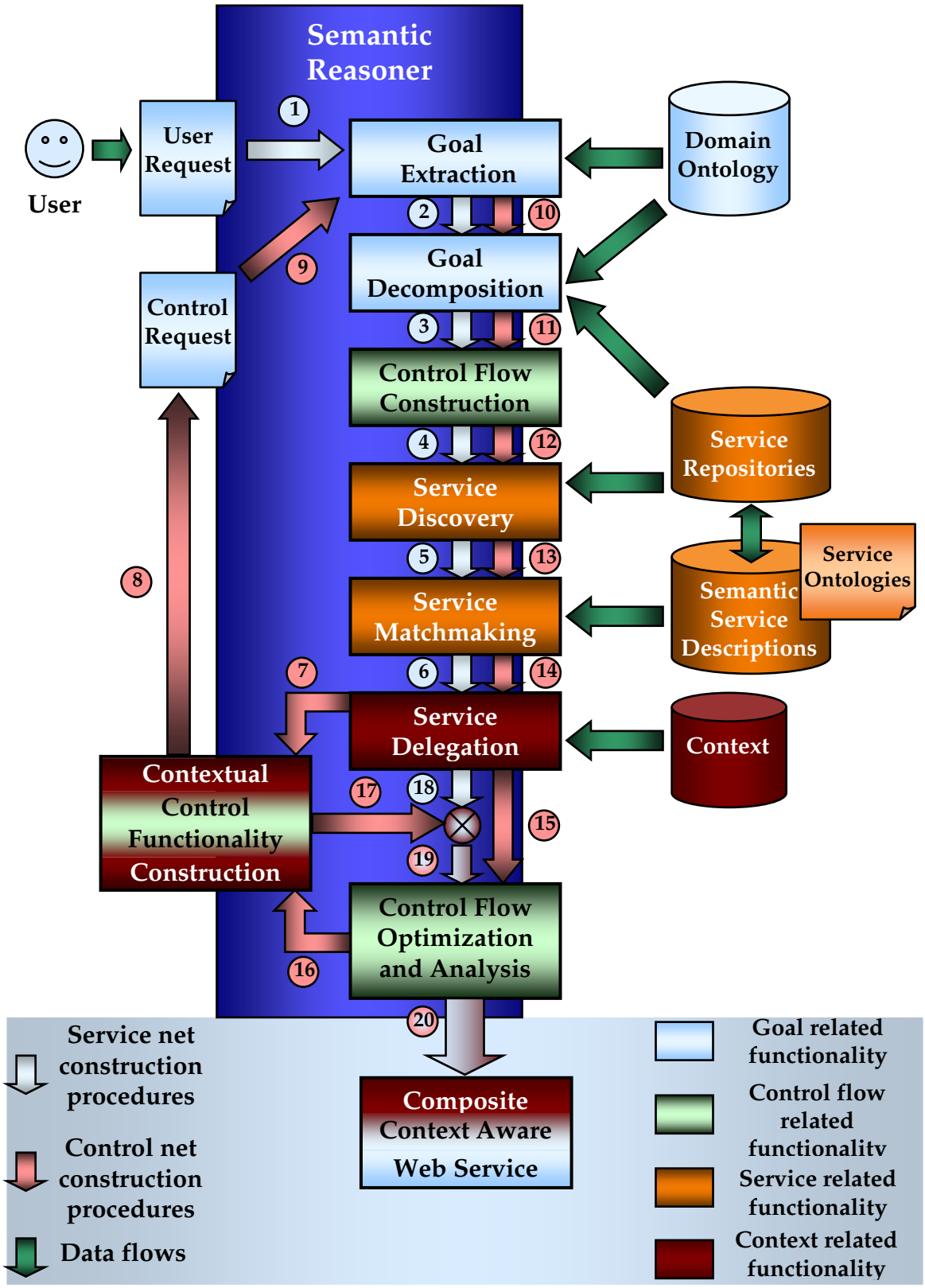

FIGURE 78 Context-aware Web service composition planning process

So, the first cycle of the planning process operation comprises the following procedures in a consecutive order:

1. Semantic composition reasoner (hereinafter SR) receives a composition request and processes it.
2. SR extracts the composition goal from the request and interprets it using respective domain ontologies.
3. SR analyzes the goal and decomposes it into primitive goals using domain ontologies. SR creates the goal structure, i.e., goal-oriented plan.
4. SR maps the obtained goal-oriented plan into a tentative generic control flow structure.
5. SR looks up available Web service repositories and discovers services which conform to the parts or the whole goal-oriented plan.
6. SR analyzes syntactic and semantic properties of discovered services using semantic service descriptions and selects appropriate services to be put into the composition plan.
7. For each service SR checks contextual requirements, which can be sensibly verified prior to execution, and discards services that do not fit into the task context. If the obtained plan still contains alternative services SR proceeds to the step 8. Otherwise it completes the control flow construction and proceeds to the step 19.

The second cycle includes the following steps:

8. If the service delegation procedure is unable to resolve some of the existing choices of alternative services on the step 7, it initiates the contextual control construction cycle. If the service delegation procedure could not check some of the services' contextual restrictions due to their dynamic nature, SR proceeds to the step 9. Otherwise, SR goes to the step 18.
9. SR issues a composition request for provision of the necessary context provider service to allow real-time verification of certain contextual restriction, which is impossible to check in advance.
10. Same as 1, but without the necessity for processing, since the request is created automatically by SR itself.
11. Same as 2, but for a different request.
12. Same as 3.
13. Same as 4, but control flow is constructed for the context control layer.
14. Same as 5. SR is looking for appropriate context provider services.
15. Same as 6. Inappropriate context providers are discarded.
16. Remaining alternative context provider services are chosen based on their non-functional properties (e.g. QoS parameters).

17. Control flow structure on the context control layer is analyzed, optimized and verified.
18. Context control layer construction is accomplished.

After the finalization of the second cycle the whole process can be finalized with the following steps:

19. Service layer and context control layer (if present) are merged into a single control flow. Composition plan is completed.
20. Context-aware composition plan is analyzed, optimized and verified, and finally implemented as a composite context-aware Web service, which is ready for execution.

The procedures constituting the composition planning process will be described in detail in Section 4.3. It must be noted that we have no intention to thoroughly model planning procedures in this thesis, since we aim to apply the generic approach to composition planning with rather small extensions and modifications to it. Neither have we intended to describe the inherent structure and functionality of semantic composition reasoner here. Instead, we will pay greater attention to the aspects bringing context awareness into the composition planning process. Also we will concentrate on developing appropriate Petri net based operational semantics for composition planning.

4.2.3 Context-aware Composition Reconfiguration

The composition reconfiguration stage potentially can be initiated as soon as the plan of the corresponding service composition is completed. The reconfiguration actually takes place during the composite service's execution. So, the composition reconfiguration stage is the run-time phase of the service's lifecycle.

The crucial moment when speaking of composition reconfiguration stage is that this stage is only possible in case of a context-aware composite Web service. When dealing with ordinary composite Web services, no reconfiguration can be performed. Ordinary composite Web Services simply enter execution stage, during which they are progressing deterministically with respect to the predefined composition plan constructed on the planning stage.

An interesting remark is that "composite context-aware service" and "context-aware composition of services" is not conceptually the same. Context-aware composition does not necessarily imply that the obtained composite service is context-aware. It only indicates that the obtained composite service has been constructed in a context-aware manner, which in our framework means that some context has been used at the planning stage to construct a fine composition plan of the composite Web service. However, context-aware composite service is somewhat more than a composite service that is constructed in context-aware manner. Context-aware composite service is supposed to include some specific

context-aware functionality, which is the valid full-fledged part of the service and is engaged during the service's execution. Thus, we can see that our Context-aware Web Service Composition framework allows both context-aware construction of ordinary composite services, and construction of context-aware composite services. Nonetheless, only context-aware composite Web services can be subjected to a reconfiguration procedure, or, in other words, only context-aware Web services are reconfigurable in our problem setting.

In case of context-aware composite service it can reconfigure its structure during execution. More specifically, it reduces its structure via removing from it certain alternative execution paths. Alternative execution paths within a composite service correspond to two or more component services, which are alternative to each other and are linked together by the operator of controlled mutual exclusion. The result of such removal is an unambiguous deterministic sequence of service operations to be executed. However, the composition plan is non-deterministic since all the choices reducing the service structure are made at run-time. Such run-time reduction of the composition structure is achieved by exploiting an extra dimension along which the composition may vary – the context. Moreover, here we specifically mean dynamic context that is acquired at run-time and is basically useless and meaningless if acquired prior to the actual execution of the service. As already stated in Section 4.1, each context-aware composite Web Service comprises one or more specific context control functions. Each such function is associated to a certain mutual exclusion operator within the control flow structure of the service. A context control function acquires context when the service is running and performs (partial) semantic match of the corresponding component service's contextual preconditions with the recently acquired context. Based on the result of this matching operation the context control function then performs reduction in the composite service's control flow structure removing all alternative paths, produced by the corresponding mutual exclusion, except one.

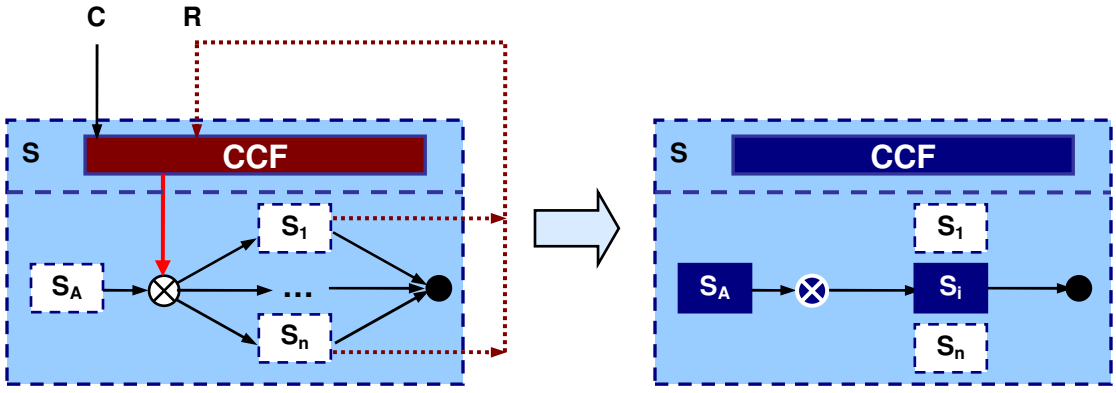

FIGURE 79 Composite context-aware Web service before reconfiguration and after it

Figure 79 illustrates a composite context-aware service's structure before reconfiguration and after it.

As it can be seen from the right-hand side of Figure 79, the reduction of the service's control flow structure essentially consists in removing the links to alternative services subjected to the operator of mutual exclusion. As a result of this action only one link heading from the mutual exclusion operator to the chosen alternative service remains, transforming the mutual exclusion operator into an ordinary sequence of two services. According to Figure 79 this reconfiguration action is instantiated as soon as the corresponding exclusion operator becomes activated, i.e., when its preceding service S_A has completed its operations. On the right-hand side of the figure (in dark blue) are shown operations which completed during the service execution, while the discarded operations are left without changes (services S_1 and S_n). Regarding the given example the performed reconfiguration procedure can be described by the following formula expressed in our service algebra:

$$S = S_A \triangleright \left(S_1 \overset{C}{\otimes} \ldots \overset{C}{\otimes} S_n \right) \overset{CR}{\Longrightarrow} S = S_A \triangleright S_i \tag{4.8}$$

where $\overset{CR}{\Longrightarrow}$ represents contextual reconfiguration transformation.

In order to formally model the removal of service links' from the service control flow we will adopt a specific extension of the Petri nets apparatus called Meta Petri nets, since they provide direct tools for modeling layered architectures with reconfiguration of lower layers by the higher ones. For now we omit the discussion of the corresponding formalization, which we shall rigorously discuss in Section 4.4, as well as the structure of context control function models and Petri net organization of the contextual control layer.

The main merit of reconfigurable composite context-aware Web services is that these services are generally more flexible in their structure and, hence, are less failure-prone. They are more reactive to changes occurring in outside world and can potentially adjust themselves to unexpected circumstances in more or less adequate way. Finally, they can also adjust their quality since we treat services' non-functional properties and, in particular, QoS parameters as context and anticipate possibilities of their use within decision making procedure for control flow reconfiguration.

4.3 Goal-Driven Context-aware Service Composition Planning

It has been shown in the Chapter 3 how to syntactically build service compositions, optimally configure them using algebraic properties of composition operators and, finally, verify their correctness prior to actual implementation. This section aims to

discuss semantic aspects of service composition: formulation of composition goals, choice of component services, considering context for increasing service appropriateness, etc. These and some more procedures constitute the service composition planning stage of composite service provisioning, which we briefly described in Section 4.2.2. Goal orientation, automation and on-demand activation are closely correlated properties of our service composition framework. Composition goal is what reflects a newly emerged user demand. On the other hand, only a well-formulated composition goal allows performing service composition in an automatic manner still ensuring composition quality, consistency and conformity with the initial user demand.

The main idea of the goal-driven automatic service composition planning can be described as follows. Any specific user demand can be generally transcribed into a formally described goal, provided that we have a formal goal description language similar to GDL4WSAC [60]. Such language should provide a comprehensive set of means for description of goals with respect to the established goal ontology and a wide variety of domain ontologies. Then the goal described using this language is processed by a special semantic reasoner and as a result decomposed into more primitive goals. Based on the determined subgoals and their relationships the engine then constructs a composition plan reflecting the composition structure but lacking any concrete functionality. This plan, which we also call "generic service", is then transferred into the corresponding Petri net representation and, finally, populated with candidate services using the "modeling from scratch" method partially described in Chapter 3. After that, composition is analyzed, verified and ready for implementation. Let us illustrate this complex procedure with the following example.

Let's assume that there is a user who is accessing a certain network via a certain mobile terminal. At some point in time the user produces a request for a certain service. (S)he may do it either explicitly (e.g. giving a corresponding command within his terminal) or implicitly (e.g. putting corresponding information into his organizer application). This request should be formally described, for example, using a dedicated goal description language so that an unambiguous goal can be extracted from it. For example, the user puts into her/his PIM application the information that (s)he is participating in a conference; which is held, say, in London on May 25th-27th, 2007. This means that the user has to make a trip to London on 24th of May, stay in a hotel there for two days and travel back on 28th of May. Obviously, the user can make all the arrangements for this trip her/himself: calling to a travel agency, putting the request for a trip, giving her/his personal data, indicating her/his preferences and limitations concerning the trip, and paying. However, the user knows that on the Web there are services which can do all this annoying stuff on her/his behalf and without her/his assistance. So, (s)he just forgets about the trip for the time being.

In the user's terminal there is a personal agent, an intelligent application that acts on behalf of the user within the electronic media accessible from the terminal. The agent constantly monitors the activities the user performs with the terminal and when using the terminal. The agent discovers that PIM application has received a new record. It checks the record and discovers a new goal for it to achieve. This goal is to make a trip.

4.3.1 Goal Extraction

As we already mentioned in Section 4.2.2, the goal concept plays a central role for the process of automated service composition planning. The composition reasoner is supposed to take a composition goal as input in order to successfully produce the required composite service output of the composition process. Since the goal is used by the reasoner automatically it must be correspondingly and sharply defined in a machine understandable format. We give the following definition of a composition goal.

Definition 4.3. *Composition goal* is an explicit and formally described specification of a desired state of affairs which has to be produced by a composite service resulting from the goal-driven composition process.

Composition goal is explicit in the sense that it is made available to the reasoner from a composition request for further processing. Its formal description is made in the format that is supported by the reasoner and allows analyzing the goal's internal structure and properties. For example, a composition request can be placed into the system as a piece of XML code describing the corresponding composition goal. An exemplary XML annotation of the composition request for a goal "make trip" is presented in Figure 80.

Figure 80 illustrates the simplest way of goal specification. A user simply enters all necessary information associated with the trip into the corresponding form provided by its PIM application's GUI. Thereafter this information is automatically encoded into a respective XML request.

The completed request is consequently directed to the composition reasoner for processing. On receiving the request the reasoner has to analyze the obtained markup and extract the composition goal from it. When we speak about composition goal extraction we assume the semantic specification of the goal is extracted from the request. Semantic extraction means that the reasoner not only should take the data from the corresponding markup, which it initially treats as an unintelligible sequence of symbols, but it should also acquire the knowledge about the goal these data describe. Acquiring knowledge basically means that the reasoner must "understand" what the received goal is. In order to do that, the reasoner should look up for appropriate domain ontologies and interpret the goal and its attributes based on them. It is not our intention to get involved into a deep

investigation of such interpretation aspects, and neither do we intend to describe the logic of the reasoner's functionality.

Whether the process of semantic goal extraction is really time-consuming or not, it always implies that some extra time is spent on finding appropriate domain ontologies and matching the goal-related data to them. It is much easier and faster for the reasoner to have a semantically annotated composition goal straight within the request. Such semantic request can be easily and automatically generated if the application that generates it is a Semantic Web enabled application, i.e., it produces and manipulates all the data with respect to a certain external vocabulary, which is commonly referred to as ontology. In the case of goal formulation this ontology is a goal ontology.

```
…
<request>
  <goal>
        <attributes>
              <name = 'make trip'/>
              <type>round trip</type>
              <starting date>
                    <day = 24 />
                    <month = 'may'/>
                    <year = 2007 />
              </starting date>
              <ending date>
                    <day = 28 />
                    <month = 'may'/>
                    <year = 2007 />
              </ending date>
              <origin>
                    <city>Jyväskylä</city>
                    <country>Finland</country>
              </origin>
              <destination>
                    <city>London</city>
                    <country>United Kingdom</country>
              </destination>
              <vehicle>plane, train, bus</vehicle>
        </attributes>
  </goal>
</request>
…
```

FIGURE 80 Extract from XML code of composition request

For example, a PIM application within the user's mobile device can be semantically enabled with the goal ontology for building goal descriptions and several domain ontologies at its disposal. These ontologies may possibly describe such concepts as trip, meeting, class, appointment, etc., allowing the application to generate semantically annotated requests for planning trips, meetings, classes and appointments respectively. The benefit of this is straightforward: the reasoner no longer has to perform time-consuming procedures of ontology search and goal interpretation. Instead, when the request is received the reasoner straight off knows which ontologies it needs to employ to interpret the goal. Moreover, it no longer needs to perform matching between the goal attributes and various properties of the ontologically described goal since within the semantically annotated request the goal attributes fully correspond to the properties within the goal ontology and the attributes' values are already described with the vocabulary dictated by the domain and range constructs within the respective ontology. Though this is a simplified view on the process of building goal descriptions using ontologies, we think it is sufficient to understand the main technical principles of goal description on the scale of this thesis.

As soon as the reasoner "realizes" the "meaning" of the established goal, it must have a comprehensive view of this goal (depending on how well the corresponding domain ontology is designed) and can start the goal decomposition process. Since the goal has now the binding to the corresponding ontology, this ontology can be taken as the reference point for the decomposition procedure.

4.3.2 Goal Decomposition

Goal decomposition procedure is of paramount importance for ensuring quality and rigor of the future service composition plan. This procedure guarantees that none of the possible courses of actions leading to the goal fulfillment is neglected or avoided at the stage of generic service composition planning. This can be generously repaid at the stage of service execution because thorough goal decomposition may consequently result in a more reliable and flexible service reconfiguration on the fly.

The idea of goal decomposition is essentially in gradual splitting of the composition goal into more and more primitive goals until elementary goals are discovered. An initial composition goal is basically a complex goal and rarely an elementary one. So, it can be basically decomposed into two or more simpler goals, which together constitute the main goal. These simpler goals are often also decomposable into yet more primitive goals and so on. Ideally, such decomposition process terminates when all complex goals are decomposed, i.e. elementary goals are reached, and the overall composition goal can be assembled exclusively with these elementary goals.

By elementary goal we understand such a goal which cannot be further decomposed into simpler goals. Even though this definition is self-explanatory, a rather philosophical issue can be raised arguing that elementary granularity of any imaginable conceptual structure can be scarcely reached. Indeed, if we attempt to imagine even some routine everyday goal and try to decompose it into more and more primitive actions to be taken, we hardly ever stop going far beyond such actions as "go there", "say that" or "take this". We can easily proceed to specifying the movements of our legs for "going there", our lips for "saying that" and our fingers for "taking this". And this level of granularity won't be probably the threshold for our imagination and decomposition. However, in real life scenarios we usually keep the granularity of our plans at the level of "go there", "say that" and "take this", or even at a more general level. Why so? Not being the experts in human psychology, we, nonetheless, believe that humans somehow balance the granularity of their actions with the level of transparency. By transparency here we mean that some actions, such as, for example, finger movements are transparent to human consciousness for a variety of tasks such as, for example, putting signatures or dialing a number with a phone. For such classes of tasks finger movement is an implicit type of functionality, which is performed on some psycho-physiological level and hardly controlled by human consciousness.

Similarly, a certain granularity level has to limit the goal decomposition process when we speak about automated systems. We see that the justification of the granularity level choice for our framework's purposes can be done by the same principle which we used for the description of human thinking. More specifically, an elementary goal in terms of goal decomposition process is a non-transparent goal that cannot be further decomposed into non-transparent goals. We define a *transparent goal* to be a goal which is not exclusively accomplishable by any known standalone piece of functionality, i.e., in terms of Web service composition there is no such service that could accomplish this goal and only this goal. If we do not place such a restriction onto the goal decomposition process, we might end up with hardware micro-operations such as binary code exchange between CPU registers in the role of elementary goals.

We will further use the term *subgoal* to encode a goal obtained as the result of goal decomposition process, be it a subgoal of the initial goal, or a subgoal of other subgoal. This will help us represent the entire goal structure as a partially ordered hierarchy based on the "ancestor-descendant" relationship. In this way we can model every non-elementary goal as a combination of its subgoals. A model of a goal that consists merely of its direct subgoals can be called *first-order* model of the goal. A second-order model of the goal can be obtained from its first-order model by inserting, instead of subgoals, their first-order models. The process of such enclosure continues recursively until all non-elementary subgoals are substituted with their first-order models. A goal is considered thoroughly modeled if it is

described with the model of the highest possible order, i.e., its model combines only elementary goals.

Figure 81 illustrates first-order models of the initial goal "make trip" and all other non-elementary subgoals of the initial goal from our previous example. Modeled goals are painted in light blue, while their subgoals have a dark orange color. It can be clearly seen from Figure 81 that first-order models (as well as higher order models) of goals describe the goal structures not only in terms of discovered subgoals, but they also identify certain relationships between those subgoals. The basic relationships between the subgoals are sequence, concurrency and exclusion. They are depicted in figure with circled red arrows, pluses, and crosses respectively. We will return to this aspect of goal modeling in Section 4.3.3.

In Figure 82 the final model of the goal "make trip" is presented. In this figure the modeled goal is painted in light blue, while dark orange color is associated with elementary goals. We can easily see that the final model of the initial composition goal contains only elementary subgoals, which is the result of recursive insertion process of the first-order models depicted in Figure 81 into one another.

Such goal decomposition procedure has a lot of resemblances with the known HTN (Hierarchical Task Planning) AI planning technique [105]. HTN planners decompose their tasks in a very similar way by recursively splitting obtained subtasks into smaller subtasks using a set of decomposition methods until primitive subtasks are reached. The major difference between this planning method and our method is that HTN planners rely on the explicitly given partially ordered set of tasks rather than on a goal formula, which is commonly used in classical planning. The tasks are basically manually defined as compositions of other tasks, which hardly allows referring to this method as a full-automated planning approach.

Even if the tasks are not manually predefined, but are acquired from the corresponding services' descriptions, appropriate services for composition are a prerequisite for HTN planners. This means that the user has to somehow compose available services manually in advance.

In contrast, we strive for a rather classical goal-driven composition approach, though we exploit the goal decomposition principle similar to HTN. So, the description of a planning domain basically consists of the initial state description and of the goal along with the set of feasible planning operators and the set of available decomposition methods. As soon as the composition goal is extracted and interpreted by the planner using appropriate domain ontologies, it can be further decomposed into subgoals using the information from the same set of domain ontologies and from a specific goal ontology, if such ontology exist. The latter option is beneficial in terms of quality and performance, since the planner does not have to spend time for searching appropriate ontologies and for

sophisticated reasoning over possibly considerable number of partially appropriate ontologies.

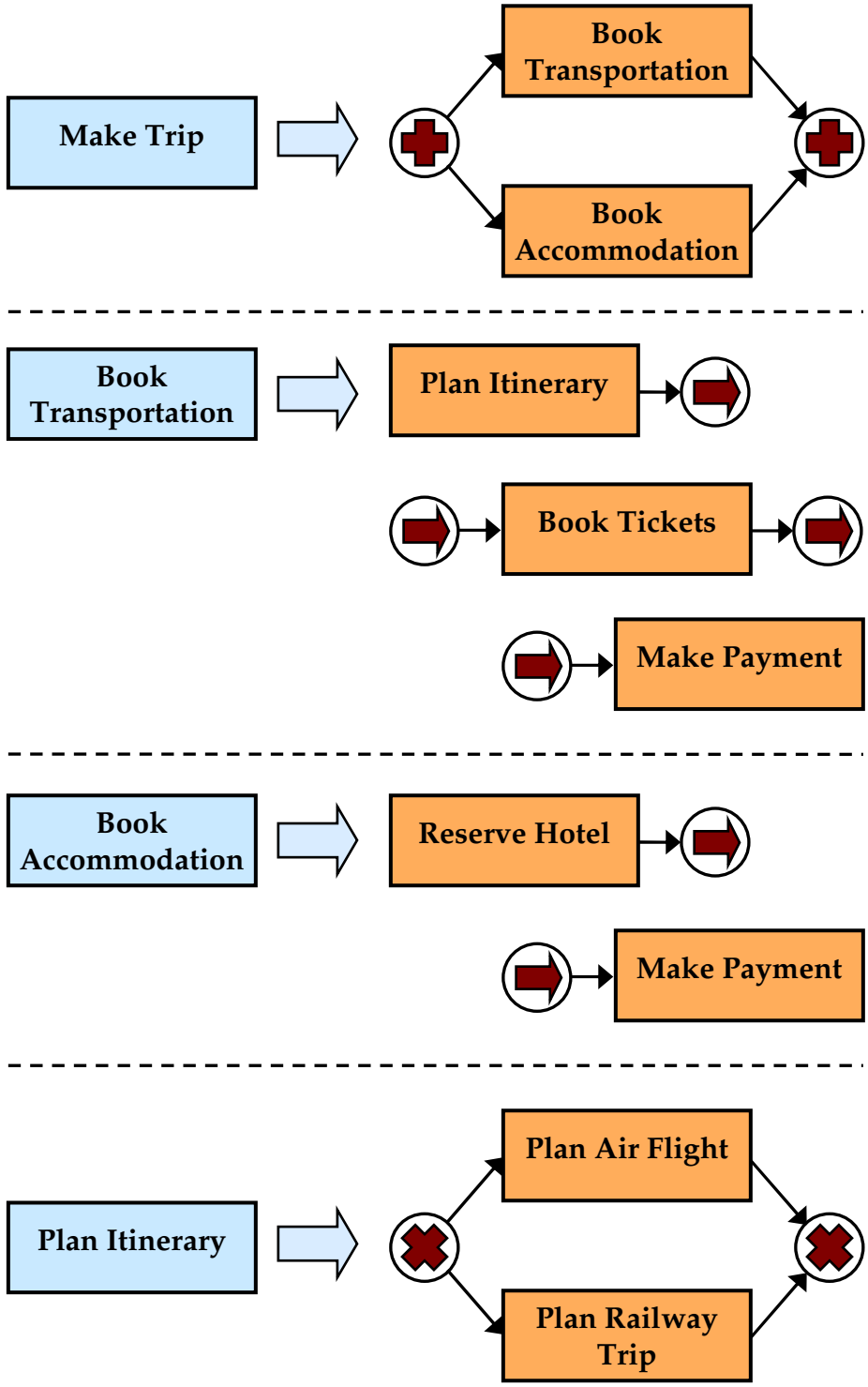

FIGURE 81 First-order models for the goal "make trip" and its subgoals

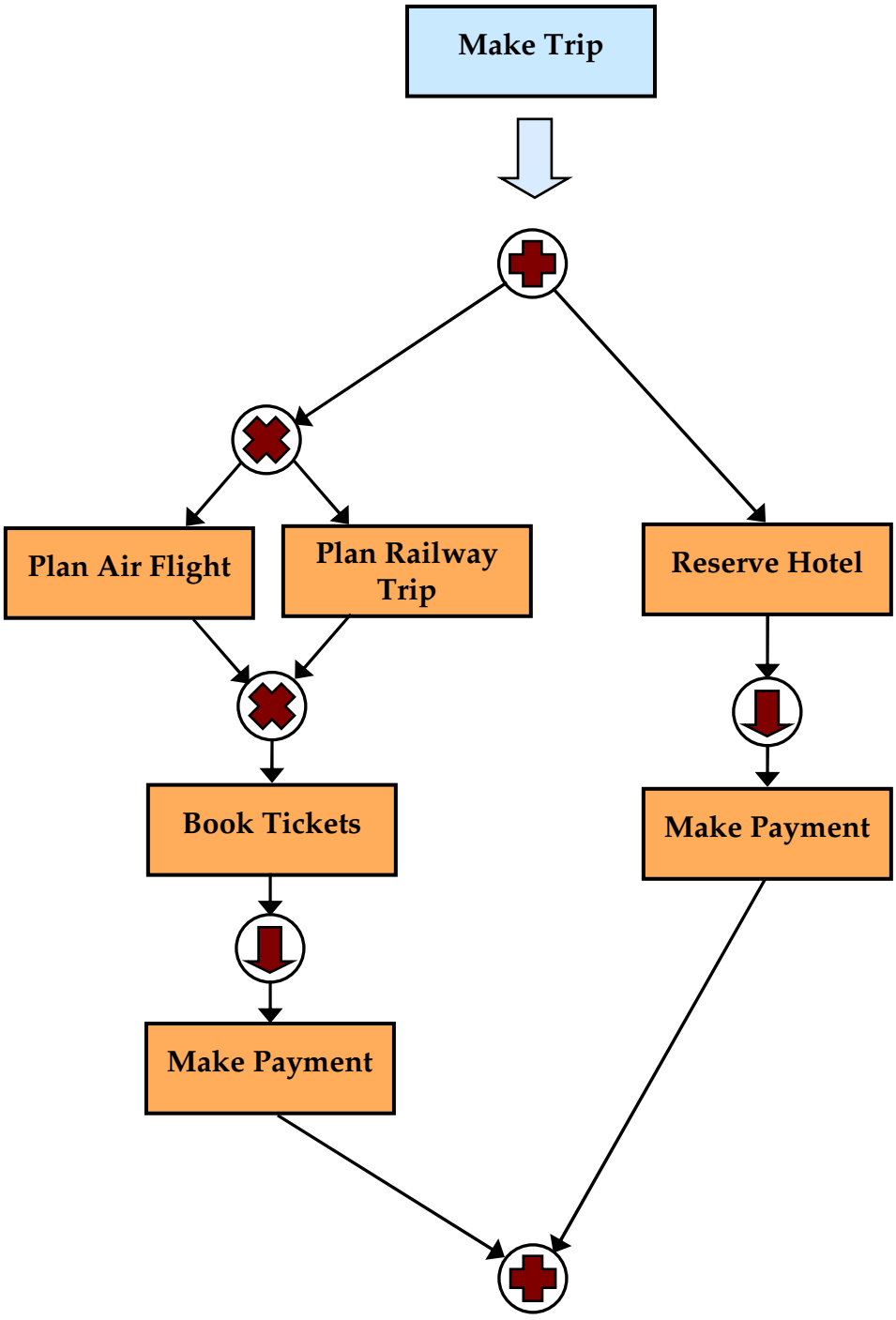

FIGURE 82 Final model of the goal "make trip"

Instead, the reasoner immediately has the comprehensive description of the goal in the form specified by a dedicated goal ontology and can instantiate decomposition

with no delays. Moreover, such ontology, if neatly designed, would not contain any uncertain, contradictory or in any other way imperfect information, which is likely to be a serious bottleneck when reasoning over multiple more general domain ontologies. However, in spite of the fact that introduction of goal ontologies would be of invaluable help to automatic semantic composition planners, their presence cannot be required as the must. The reason is that the planner would require manual creation of a new goal description, whenever the appropriate goal ontology is absent, which makes the fully automated nature of such composition planning at least questionable.

Nonetheless, we believe that goal decomposition still can be based on available domain ontologies if they comprehensively describe the respective planning domain. We do not intend to describe any precise details on such ontology analysis, since it is perhaps a fairly large problem to cover, and it is not claimed among the contributions of this thesis. We only note that such goal decomposition procedure can be significantly enhanced by considering appropriate service ontologies. More specifically, if the goal cannot be decomposed adequately enough on the basis of domain ontologies, and since the goal being decomposed has been already extracted and interpreted, the reasoner can discover services which achieve this particular goal, acquire their service ontologies (written in OWL or WSML), and finally devise an appropriate decomposition based on their semantic descriptions. This type of decomposition is but a worst case scenario. Its main drawback is the improbability of creating universal goal decomposition with respect to highly specific service structures. In any case, it is definitely better than nothing. It should be also noted that such enhancement of goal decomposition procedure implies some efforts on service discovery, which is in fact to be made on later stages of frameworks operation. So, goal decomposition and service discovery can be partially interleaved in some cases, which may be beneficial from the perspective of later service discovery under worst-case scenarios.

The described goal decomposition procedure operates in multiple iteration cycles. During each cycle it decomposes a subsequent subgoal as illustrated in Figure 81. As soon as all decomposable subgoals are decomposed the procedure assembles the final model of the initial composition goal as shown in Figure 82. Thus, a goal-driven generic composition plan is built.

However, we still have to make a couple of essential remarks regarding the limits of decomposition deepness and the structure of the final model of the composition goal.

We already indicated above in this section that the level of decomposition granularity, i.e., the deepness to which the decomposition process is drilling down has to be controlled. We also proposed to use available service functionality as the measure of the smallest elementary subgoal of any goal. This leads us to the idea that at each cycle of decomposition for each of the obtained subgoals it must be checked whether there exist ready-made services which exclusively perform the

corresponding subgoals. And if such services can be discovered, decomposition process is allowed to proceed to the next level. Otherwise, if at least one of the obtained subgoals has not found a match among available services, this particular subgoal and the whole level of its super-goal descendants are discarded, and decomposition is stopped. In case of applying such decomposition control facility, service discovery of potential component services will be largely done in parallel to goal decomposition. The benefit of this is obvious: decomposition of each subsequent goal can be performed on the base of the description of the service that was discovered on the previous level of decomposition when this particular goal was obtained and the corresponding service was discovered to verify that this goal is not yet an elementary one. Thus, service discovery efforts made on the stage of goal decomposition allow killing two birds with one stone: ensuring that decomposition can be soundly made, and verifying that it is actually needed within the composition plan because there are services implementing the subgoals obtained as the result of decomposition. Furthermore, in such application, service discovery efforts are no longer extra efforts, since at service discovery stage only services alternative to the services discovered during goal decomposition stage must be discovered. Thus, service discovery is partially shifted to the stage of goal decomposition.

Another interesting observation concerning the granularity level and the control facility proposed in the previous paragraph is that we can easily come across a situation where a certain goal does not have a service counterpart, but its subgoals do have such counterparts. According to the principle described above such a goal will be immediately discarded by the planner, though it is obvious to us that it is not elementary. As a special case of this we can admit that the initial composition goal does not have a service counterpart, which is quite probable because naturally composition request demands functionality which is not otherwise available. Because the planner works recursively this goal will be also immediately discarded, which is apparent nonsense. We find this unanticipated problem to be difficult to formalize and to cope with. However, a rather simple solution suggests itself. This solution dictates that no goal should be proclaimed elementary if its first-level subgoals do not have service counterparts. Instead, the planner must remember a certain default integer number that denotes the number of decomposition levels to be checked at maximum before proclaiming a goal elementary. To be more specific, say, this number is 3. The planner starts decomposing an arbitrary goal A, discovers its first-level subgoals B and C, and finds no services for one or both of these subgoals. Nevertheless, it does not name the goal A as elementary. Instead, it decomposes goals B and C into subgoals B1, B2, C1, and C2, and again discovers no services for at least one of these goals, which are second-level subgoals for the goal A. The planner again decomposes the newly obtained goals and obtains their subgoals, which are third-level subgoals of the goal A. If one of these subgoals again does not have a service counterpart, then

the goal A is declared elementary and decomposition is considered infeasible. Otherwise, if all of the third-level subgoals of the goal A have service counterparts, decomposition is feasible and goal A's three-level hierarchy is put to the overall model.

To clarify things even more, we now give several definitions of different goal statuses. With these statuses we will be able to more clearly explain the principle of goal decomposition.

Definition 4.4. *Potentially elementary goal* is a goal which has a valid service counterpart.

Definition 4.5. *Potentially decomposable goal* is a goal which can be decomposed on the basis of domain ontology analysis or has a valid non-atomic service counterpart (see Chapter 3 for the definition of atomic service).

Definition 4.6. *n-Decomposable goal* is a goal the subgoals of which at the n^{th} decomposition level are potentially elementary provided that the goal itself is set on the level 0.

Definition 4.7. *Decomposable goal* is a goal which is *n*-decomposable for some $n \leq N$, where N is the granularity level control constant, which expresses the number of the maximum goal decomposition level, at which subgoals are checked for being potentially elementary.

Definition 4.8. *Elementary goal* is a potentially elementary goal which is not decomposable.

Definition 4.9. *Infeasible goal* is a goal which is neither potentially elementary, nor potentially decomposable.

Now using these definitions we can sketch the algorithm of goal decomposition procedure operation. It should be noted that the procedure operates recursively, and is built with respect to the "breadth-first search" principle. The algorithm includes two nested cycles. The *i*-cycle changes decomposition levels, while the nested *j*-cycle changes subgoals within each level. It should be noted that on each iteration the algorithm processes one subgoal.

1. Planner sets $i=1$, $m=1$ and adds the initial goal G_1 to the model.
2. If $m > 0$, planner sets the number of iterations J to the number m of goals G_i^j on the same level i. Otherwise, planner proceeds to the Step 11.
3. Planner sets $j = 1$, $m = 0$.
4. If $i \neq 1$, planner checks if the goal G_i^j has an ancestor within the model. If not, planner discards the goal and proceeds to the Step 9.
5. Planner checks if the goal G_i^j is potentially elementary. If the goal G_i^j is potentially elementary, planner proceeds to the Step 7.

6. If $i > N$, planner finds the ancestors $G_{i-n}^{r}, n = \overline{1,N}$ of the goal G_i^j and sequentially checks whether they are potentially elementary or not. If none of the ancestors is potentially elementary, planner declares the ancestor G_{i-N}^{r} non-decomposable and discards it from the model as well as all its subgoals starting from level $i-N+1$ and ending at level $i-1$. Planner also discards the goal G_i^j from the model.
7. Planner checks if the goal G_i^j is potentially decomposable. If the goal G_i^j is potentially decomposable, planner decomposes the goal G_i^j into p subgoals $G_{i+1}^{m},...,G_{i+1}^{m+p}$, adds them to the model and sets $m = m + p$. Planner proceeds to the Step 9.
8. Otherwise, if the goal G_i^j is neither potentially elementary, nor potentially decomposable, planner declares the goal G_i^j infeasible and discards it and all its ancestors from the model until it finds the ancestor G_{i-k}^{l} which is a potentially elementary goal. Planner declares G_{i-k}^{l} the elementary goal. Planner discards from the model all the subgoals of the goal G_{i-k}^{l} on all the levels from starting from level $i-k+1$ and ending at level $i-1$.
9. If $j < J$, planner increments *j* and proceeds to the Step 4.
10. Otherwise, planner increments *i* and proceeds to the Step 2.
11. Planner completes the model of the goal G_1.

The presented algorithm builds a complete model of the initial goal. It is important that the algorithm guarantees that any decomposed goal can be, at a current moment, achieved by a joint operation of a set of real services, i.e., there is no goal which cannot be turned into an executable service composition because of the unavailability of appropriate component services. The only matter to be still verified is whether these services are actually composable with one another.

The algorithm relies on the granularity level control constant *N* to sensibly limit the decomposition depth. This constant must be chosen carefully. Should it be too small really complex goals would frequently be non-decomposable due to the absence of complex services that could fit as counterparts for its first-level subgoals. On the other hand, should it be too big computational complexity of the algorithm might grow significantly turning the algorithm poorly scalable as a consequence. Intuitively, we feel that 3 is a reasonable value for this constant in a majority of scenarios.

Regarding the structure of the goal decomposition models produced we need to remark that the algorithm actually produces slightly different model structures than that presented in Figure 82. Despite its simplicity, the sample model of Figure 82 does not fit well for our composition planning purposes because it represents an

initial composition goal merely in terms of the elementary subgoals. This is somewhat unsuitable since it does not allow us to include, in our consequent compositions, such services that correspond to non-elementary goals which are still valid subgoals of the initial goal. From our viewpoint, this would be a serious limitation to our composition framework. Impossibility to use component services to fill non-elementary goals would significantly reduce the flexibility and effectiveness of composition construction. Therefore, we prefer to use a different format for modeling goal decompositions, which we separately describe in the following section.

4.3.3 Goal Decomposition Modeling

As we already described above, during goal decomposition the reasoner extracts all possible subgoals of the given goal using a domain ontology or existing services. Then it similarly extracts subgoals of the subgoals. It continues this procedure recursively until it finds all the elementary subgoals and no complex subgoals remain unanalyzed. This way the reasoner acquires the models of all non-elementary subgoals such as shown in Figure 81. Using these models and the HTN planning principle, the reasoner finally constructs a *goal tree*. Goal tree is a graph: its root is the initial goal, the transient nodes are the subgoals, and the leaves are the elementary goals. This way the goal tree represents a kind of the conceptual model of the given goal. The goal tree for the goal "make trip" is shown in Figure 4.12.

After that the reasoner analyzes the additional information contained by the request and performs another matching. As a result of this procedure some subgoals (transient points) may completely vanish from the tree, with all their successors up to the leaves. In reality this means that user preferences, requirements, restrictions concerning the goal, i.e., user context, are formulated so that some of the subgoals become unacceptable in principle. For example, if somewhere in the user profile it is stated that this user suffers sea-sickness or in the goal description the user indicates "no sea travel" under the paragraph "means of travel", then the reasoner will just cut out of the tree the whole branch corresponding to traveling by sea transport as illustrated by Figure 84.

It is straightforward that the less detailed the goal description given by the user is the wider will be the goal tree generated by the reasoning engine and vice versa.

As a matter of fact, tree representation of the goal is somewhat restricted to adequately capture the real structure of the goal. In certain cases it does not give a correct view. For example, if two subgoals residing on the same level and having the same ancestor are not supposed to be achieved simultaneously and in parallel, but in such a way that one goal requires that another is achieved first, then the basic tree representation is no longer suitable for describing the goal structure.

For example, suppose we put a first-level subgoal "make payment" into our tree in Figure 83. This subgoal cannot be performed until at least one of the other two subgoals is achieved because the user must first know how much and whom to pay, not to mention that it is plain absurd to pay for nothing. This issue can be also regulated by appropriate design of ontologies. Nevertheless, for the moment and for the sake of simplicity we neglect this tricky nuance and assume that the ontologies are designed to exclude such ambiguities.

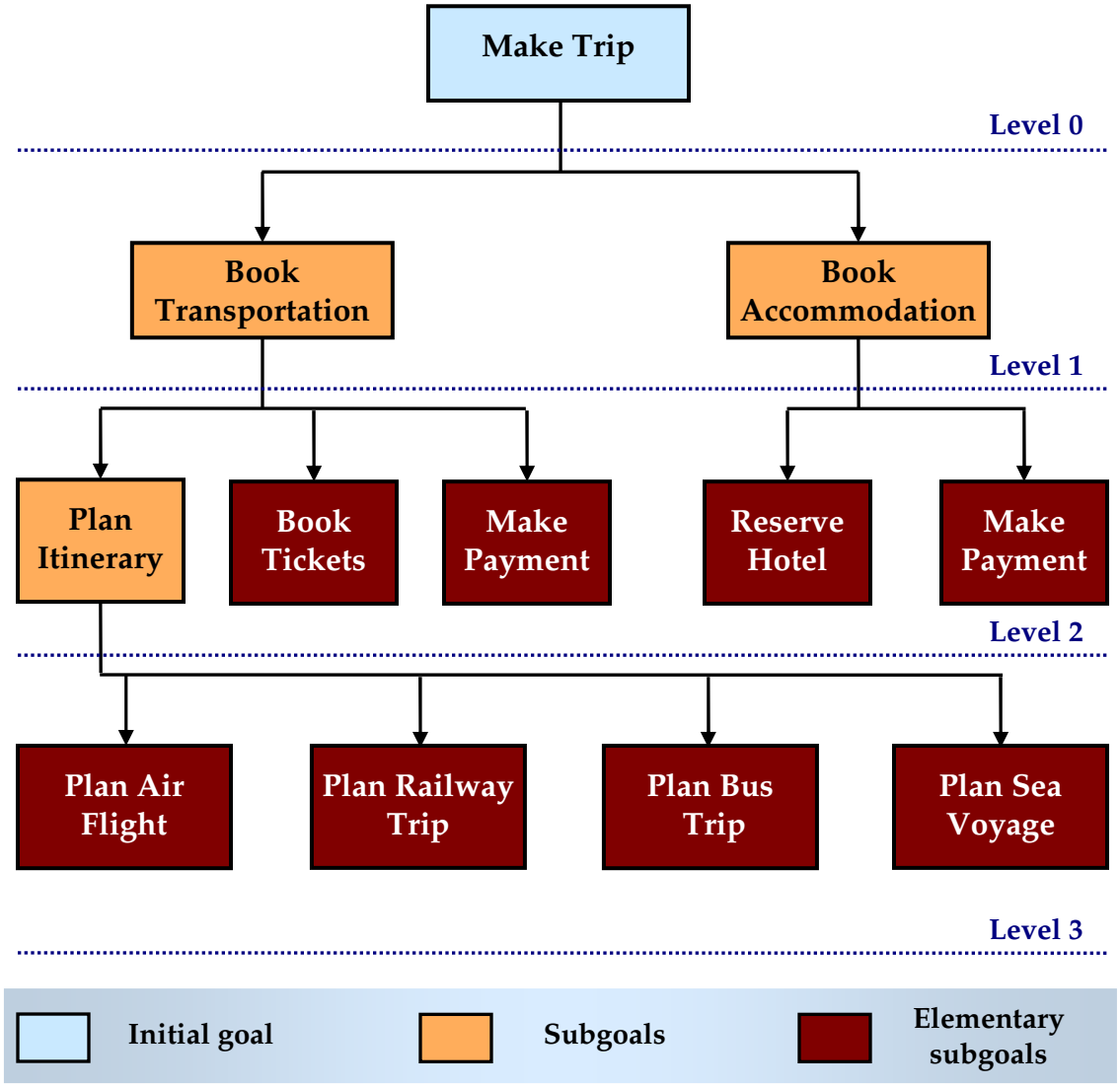

FIGURE 83 Goal tree for the goal "make trip"

However, we still have a somewhat ambiguous case where two or more subgoals are alternative or mutually exclusive. For instance, in the majority of cases a traveler uses only one travel method, and very rarely a combination of them. The user will basically prefer to go to London either by airplane or by train. This means

that two subgoals "book flight" and "book railway trip" are mutually exclusive and relate to one another by "OR" relationship. Such type of relationship cannot be formalized by the basic tree formalism. Goal trees are a good tool for describing the structure of goals, but almost useless for describing logical and causal relations between the elements of the structure. To comprehensively describe those relations, it is necessary to utilize another tool, which at least allows formalizing logical disjunction and logical implication relationships (logical conjunction "AND" is formalizable by default within the tree representation). Propositional logic fits well for such purposes, so we'll use it along with goal trees.

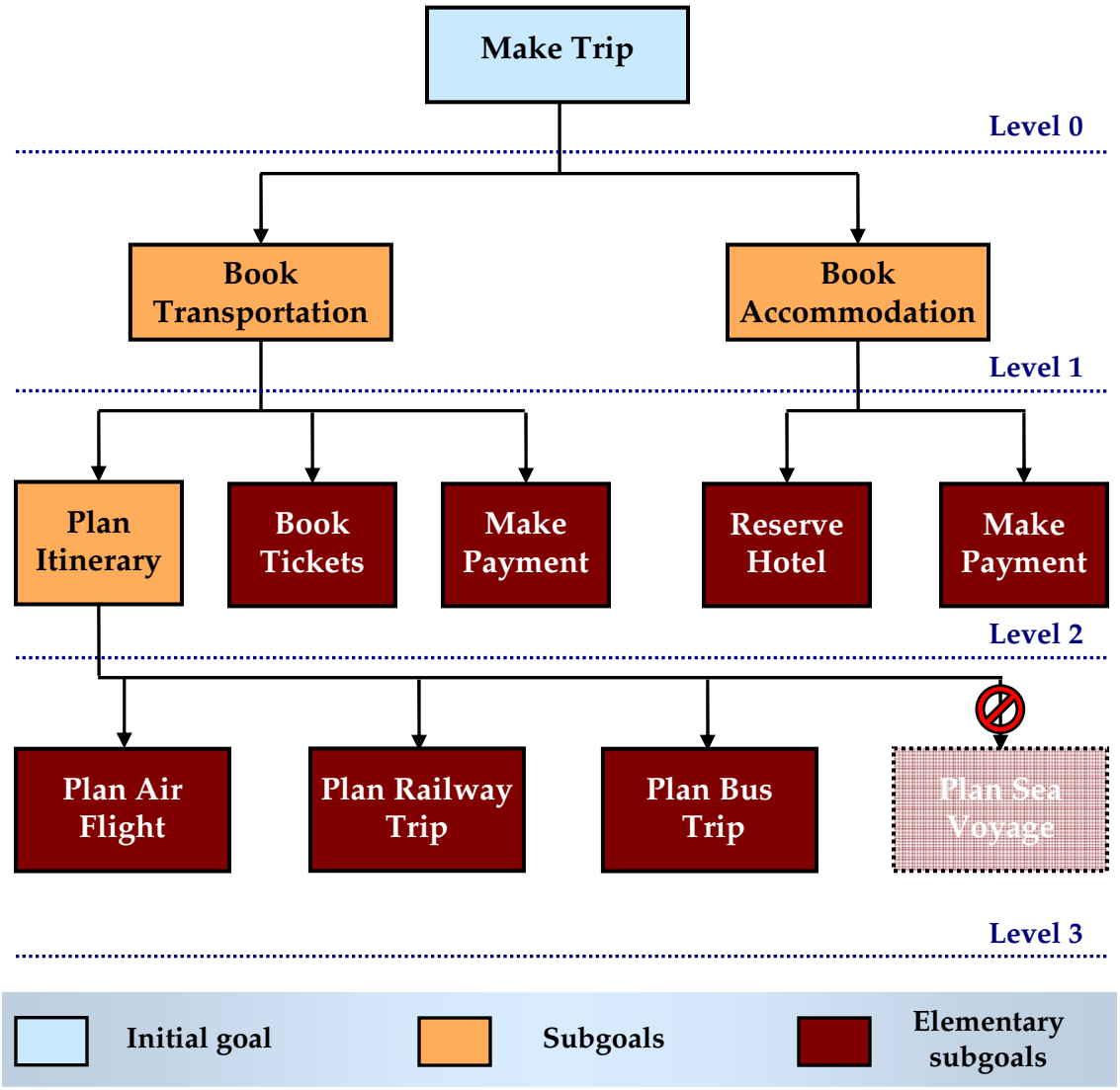

FIGURE 84 Goal tree with the restriction "no sea travels"

Let us formalize the goal "make trip" using the goal tree and appropriate description with propositional logic. The goal tree of Figure 83 does present the

structure of the goal, but it shows no relations between the subgoals. To do that, let's build an additional goal description with propositional logic. For that let's first annotate all the subgoals with propositions. We will get the following expressions:

G: Make Trip
A: Book Transportation
B: Book Accommodation
C: Plan Itinerary
D: Book Tickets
E: Make Payment (Transportation)
F: Reserve Hotel
H: Make Payment (Accommodation)
I: Plan Air Flight
J: Plan Railway Trip
K: Plan Bus Trip
L: Plan Sea Voyage

Using these literals we can now construct the following propositions describing relations between the subgoals:

$G : A \wedge B$,
$A : C \rightarrow D \rightarrow E$,
$B : F \rightarrow H$,
$C : I \vee J \vee K \vee L$,

where :

- $\wedge$ denotes the logical conjunction "AND" relation when two subgoals should be achieved both, independently and in parallel;
- $\vee$ denotes the logical disjunction "OR" relation when two subgoals are alternative and mutually exclusive, so that only one of them should be achieved;
- $\rightarrow$ denotes the logical implication relation when the antecedent subgoal should be achieved prior to the consequent subgoal.

The goal tree refined with the given propositional relations is presented in Figure 85.

Several important observations can and should be made with respect to the built goal model.

First of all, it can be easily seen that only goals residing on the same level are linked by certain logical or causal relationships. The goals residing on different levels are basically always related to each other by "subgoal of" relation if they have a direct chain of links through other subgoals, i.e. one is the subgoal of certain order for the other. Otherwise, if they only have a common ancestor, they are not exposed to this type of relation.

Another interesting observation is that goals residing on the same level are not always explicitly subjected to any logical relationships. For example, no

relationship is established between subgoals "book tickets" and "reserve hotel" within the model of Figure 85. This always takes place when two subgoals do not have a direct common ancestor. So, generally only subgoals having a common direct ancestor are linked by explicit logical relationships. However, the subgoals "book tickets" and "reserve hotel" are, in fact, conjunctive, since they inherit such relationship from their ancestors. Thus, goal tree models exhibit inheritance of logical properties. The general inheritance rule looks as follows: if two subgoals *a* and *b* residing on the same level *i* exhibit the logical relationship *R* such that $a_i R b_i$, then all the descendants a_n of the subgoal *a* relate to all the descendants b_n of the subgoal *b* such that:

$$a_n R b_n, \forall n, i \le n \le N \tag{4.9}$$

where *N* is the total number of levels within the goal model.

It is obvious that by the way we formulate logical relationships between subgoals we are aiming to match them later with the composition operators presented in Chapter 3. So, logical implication corresponds to the sequence operator, logical disjunction to the mutual exclusion operator, and logical conjunction to the parallelism operator. It may be noted that this way we model only relationships corresponding to basic composition operators. Most of the advanced operators can be obtained either through combining the basic ones appropriately, e.g., the unordered sequence operator, or by extending basic operators with certain data flow primitives, such as parallelism with communication or parallelism with resource sharing. The former case we described in Chapter 3, while the latter is beyond the scope of this work.

Speaking of the refinement operator, two things must be noted. Firstly, this operator is almost completely non-applicable when modeling purely composite services from scratch, which is exactly our case. So, our framework should not really account for this operator. Secondly, it can be easily noticed that this operator perfectly matches to the "subgoal of" relationship, which is the basis of goal modeling.

The iteration operator can be modeled as a reflexive logical relation graphically depicted as a loop over the corresponding subgoal.

The goal decomposition model presented in Figure 85 does not contain any cross-relationships. By cross-relationships we mean such binary relationships between three or more subgoals which make it difficult to determine the order of this relationships' application. In order to demonstrate a model with cross-relationships let us reformulate the goal model of Figure 85 as follows.

G: Make Trip
A: Book Transportation
B: Book Accommodation

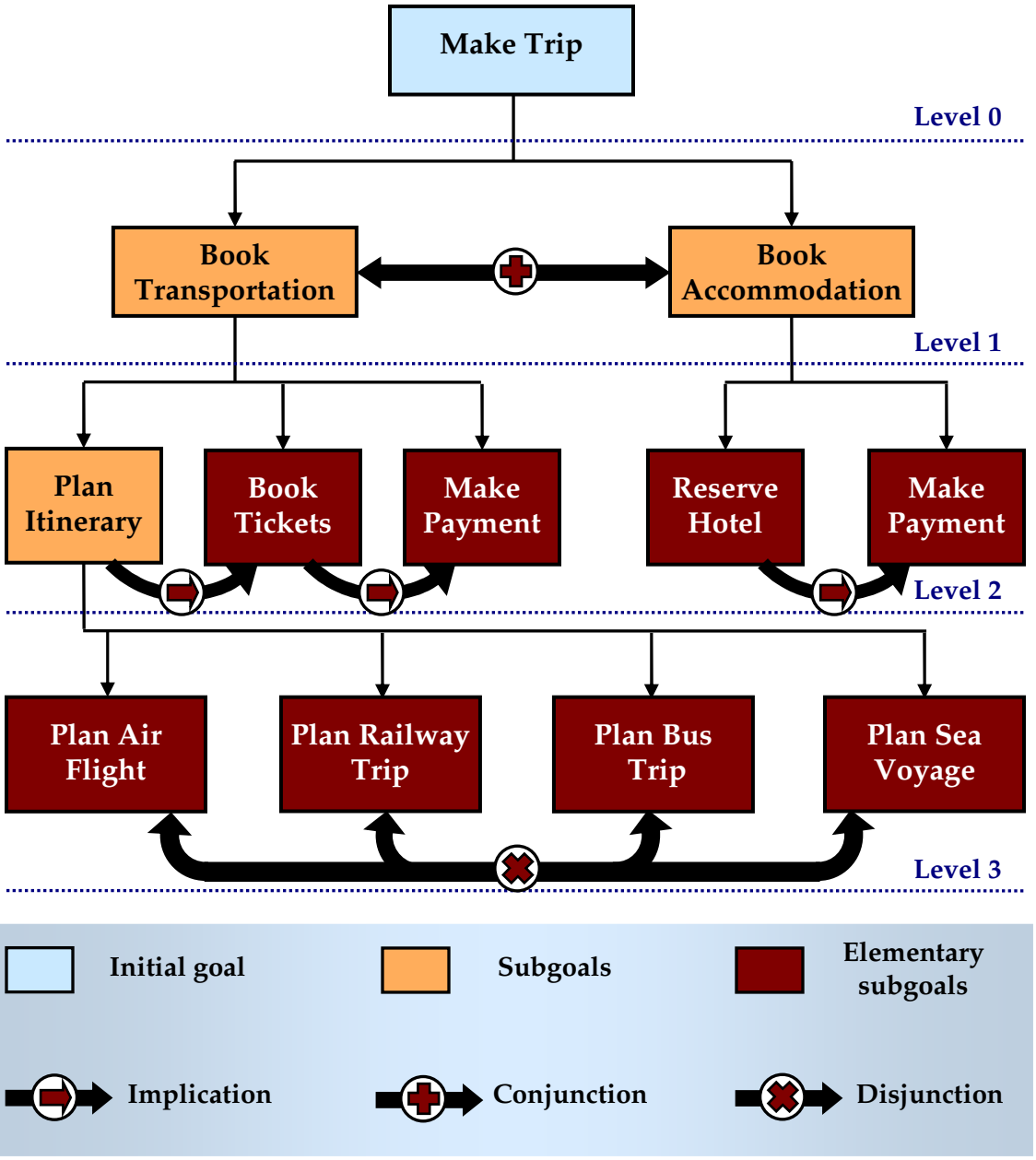

FIGURE 85 Goal tree with propositional relationships between nodes

C: Book Air Flight
D: Book Railway Trip
E: Book Taxi
F: Reserve Hotel
H: Make Payment (Hotel)
I: Plan Itinerary (Flight)
J: Make Payment (Flight)

K: Plan Itinerary (Railway Trip)
L: Make Payment (Railway Trip
M: Plan Itinerary (Taxi)
N: Make Payment (Taxi)

Using these literals we can now construct the following propositions describing relations between the subgoals:

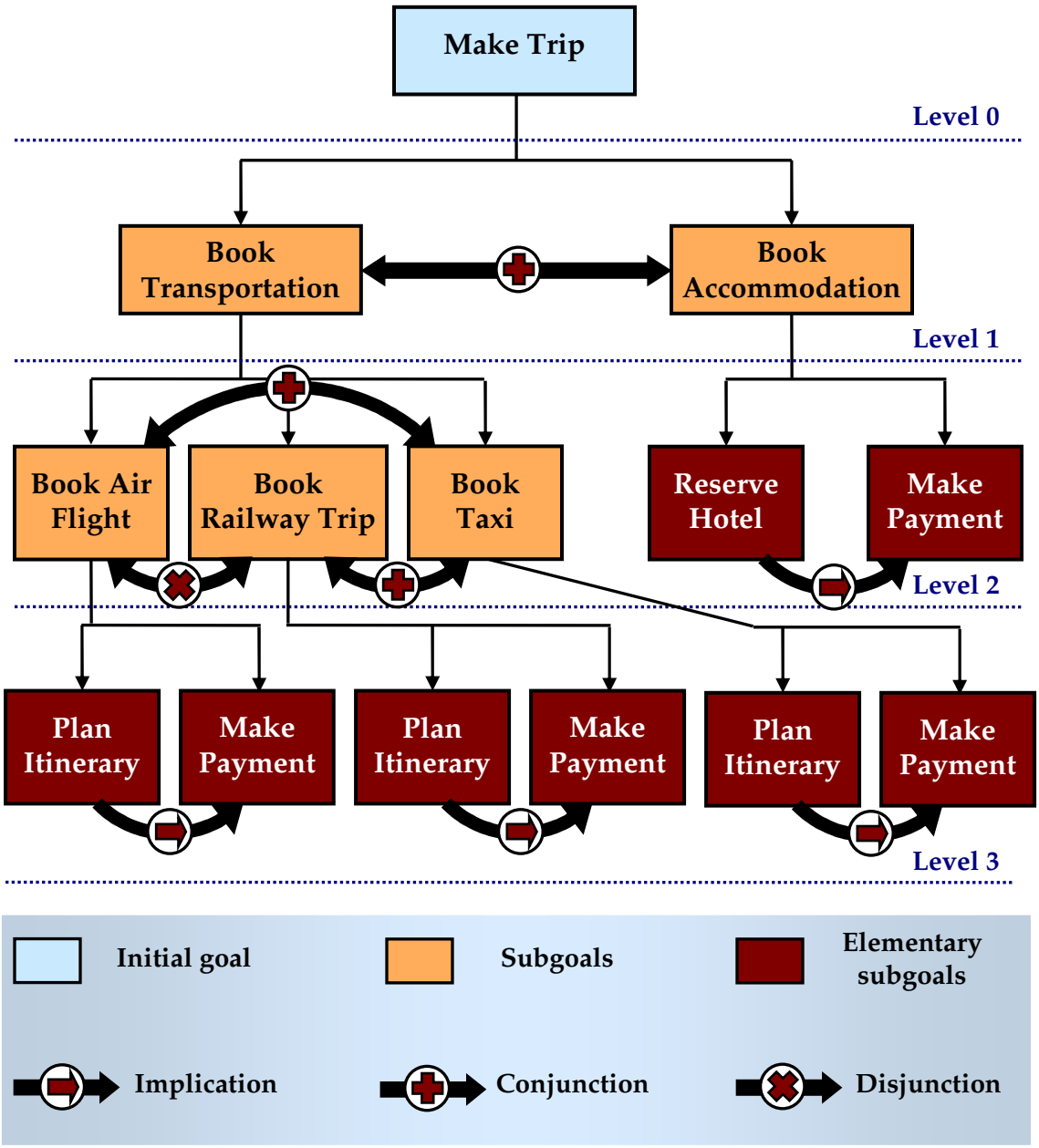

FIGURE 86 Final goal tree for the goal "make trip"

$G : A \wedge B$,
$A : (C \vee D) \wedge E$,
$B : F \rightarrow H$,
$C : I \rightarrow J$,
$D : K \rightarrow L$,
$E : M \rightarrow N$.

The corresponding goal model is presented in Figure 86. We can see that the subgoal *A: Book Transportation* has three subgoals, namely, *C: Book Air Flight*, *D: Book Railway Trip*, and *E: Book Taxi*. These three subgoals form a cross-relationship: *C* and *D* are mutually exclusive, while *E* is parallel to them. Such combination of binary relationships can be resolved: first the choice between alternatives *C* and *D* has to be made, and then the chosen option has to be performed in parallel to *E*. However, this is not clear from the model of Figure 86. The graphical goal model is incapable of distinguishing the particular order of the relationships between cross-related subgoals, i.e., has no means for prioritizing these relationships. We considered a rather simple case, where the right order can be devised intuitively and is, for example, indicated by the fact that we have one disjunction relationship and two conjunction relationships between the corresponding subgoals, so that it is easy to guess that conjunction is distributed over disjunction. However, there can be cross-relationships which are much more difficult to resolve with such a graphical model lacking the formality of interrelationships' description. Therefore, the graphical model has to be always extended with some more formal description of subgoals' relationships. We propose propositional logic expressions in the role of such formal description. For the given example the expression $A : (C \vee D) \wedge E$ describes the correct prioritization of individual relationships quite formally and unambiguously.

In this section we described in detail the format of goal models which is to be used during composition planning procedure. The goal decomposition procedure described in the previous section is supposed to produce exactly these types of goal decomposition models. If, however, the decomposition procedure fails to produce a valid goal model, it can retry to decompose a given composition goal on a different set of domain ontologies and available services. Should the goal decomposition procedure successfully produce a goal decomposition model, this model can be directly applied for the construction of the corresponding composition control flow.

4.3.4 Goal-based Control Flow Construction

As soon as the goal tree is built and refined with propositional descriptions the reasoner has to transform it into the Petri net model of the composition control flow. The process of control flow construction is based on the provided goal decomposition model and the corresponding service net is built with respect to the

given goal model. At this stage of composition planning the so-called *generic service net* is built. By generic service net we mean such service net which does not include any component services or other functional entities in its structure. The principles of goal-based construction of a generic service net are the following:

- input place i corresponds to the beginning of the goal G (the root of the tree);
- output place o corresponds to the end of the goal G (the root of the tree);
- every transient point (subgoal) K of the tree has starting k_i and ending k_o places associated with it;
- each leaf of the tree (elementary subgoal) L has the only place l associated with it;
- starting place k_i of every transient point (subgoal) K is located after the starting place j_i of its ancestor point J, i.e. k_i as an input has a transition which is the output of j_i;
- ending place k_o of every transient point (subgoal) K is located before the ending place j_o of its ancestor point J, i.e. j_o as an input has a transition which is the output of k_o;
- relation $\wedge$ between subgoals J and K is modeled by the parallelism operator which inserts:
  - a "split" transition t_{j+k} having two outgoing arcs leading to subgoals' starting places j_i and k_i respectively;
  - a "merge" transition $t_{j+k'}$ having two incoming arcs coming from subgoals' ending places j_o and k_o respectively;
- relation $\vee$ between subgoals J and K is modeled by the mutual exclusion operator which inserts:
  - two separate transitions t_j and t_k having the same "split" input place and a specific output place j_i and k_i respectively;
  - two separate transitions $t_{j'}$ and $t_{k'}$ having the same "merge" output place and a specific input place j_o and k_o respectively;
- relation $\rightarrow$ between subgoals J and K is modeled by the sequence operator which inserts one transition t_{j-k} having one input place j_o and one output place k_i.

The obtainable generic constructs are shown in Figure 87.

Now we can build a Petri net model of the goal "make trip". Using the propositional expressions describing the goal and its subgoals we can write the following goal formula:

$$G:(((I \rightarrow J) \vee (K \rightarrow L)) \rightarrow (M \rightarrow N)) \wedge (F \rightarrow H) \tag{4.10}$$

It should be noted that we intentionally changed the relationship $A:(C \vee D) \wedge E$ to the relationship $A:(C \vee D) \rightarrow E$, since it is logical that taxi at the destination site

should be booked only after the actual trip is booked and the exact destination is known. So, taxi booking depends both logically and functionally (because it requires the destination site as an input) on main transportation booking results.

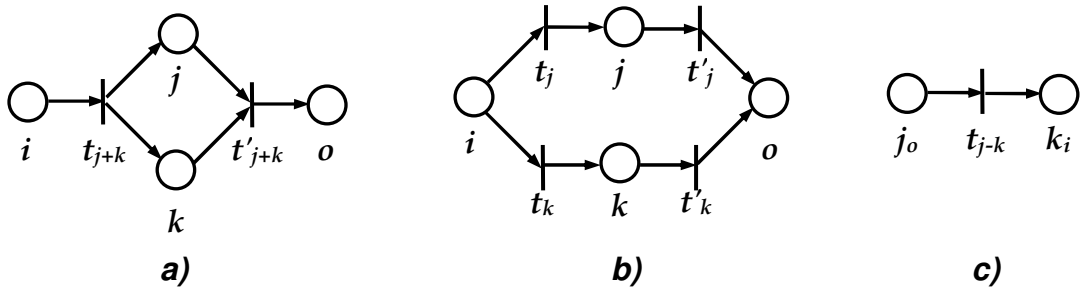

FIGURE 87 Modeling of subgoals' relationships with Petri net algebra operators: a) conjunction, b) disjunction, c) implication

Due to the above described control flow construction principle, goal formula (4.10) can be transcribed into the following expression written in Web service algebra:

$$G:(((I \triangleright J)\otimes(K \triangleright L)) \triangleright (M \triangleright N))+(F \triangleright H) \tag{4.11}$$

Formula (4.11) can be notationally reduced using prioritization of algebra operators described in Chapter 3:

$$G:(I \triangleright J \otimes K \triangleright L) \triangleright M \triangleright N + F \triangleright H \tag{4.12}$$

However, keeping redundant parenthesizing such as in formula (4.11) can be often useful. In formulae (4.10)-(4.12) the main goal G is expressed only via elementary subgoals. Non-elementary subgoals are contained within parentheses in formulae (4.10) and (4.11). But not all the parentheses express the bounds of non-elementary subgoals. Thus, the goal formula alone cannot provide unambiguous description of the goal. Neither can the goal tree do that. So, goal trees should be used together with goal formulas to ensure unambiguousness of the given goal representation. Now, using the goal tree of Figure 86 and the goal formula (4.11) we can construct the generic service net of the composite service that aims at achieving the goal "make trip".

The corresponding structure of the generic service net is shown in Figure 88. Within the built Petri net structure all non-elementary subgoals are represented by exactly one starting P_i and one ending P_o place (painted in orange), which signify the corresponding goal initiation and finalization states respectively. The initial composition goal is also associated with one starting G_i and one ending G_o place (in light blue), which have the same meaning and are simultaneously the starting and the ending place of the whole generic service net. All elementary subgoals are represented by a single place P per goal (in dark red). These places represent empty services to be filled later on with real services which aim to achieve the corresponding elementary subgoals.

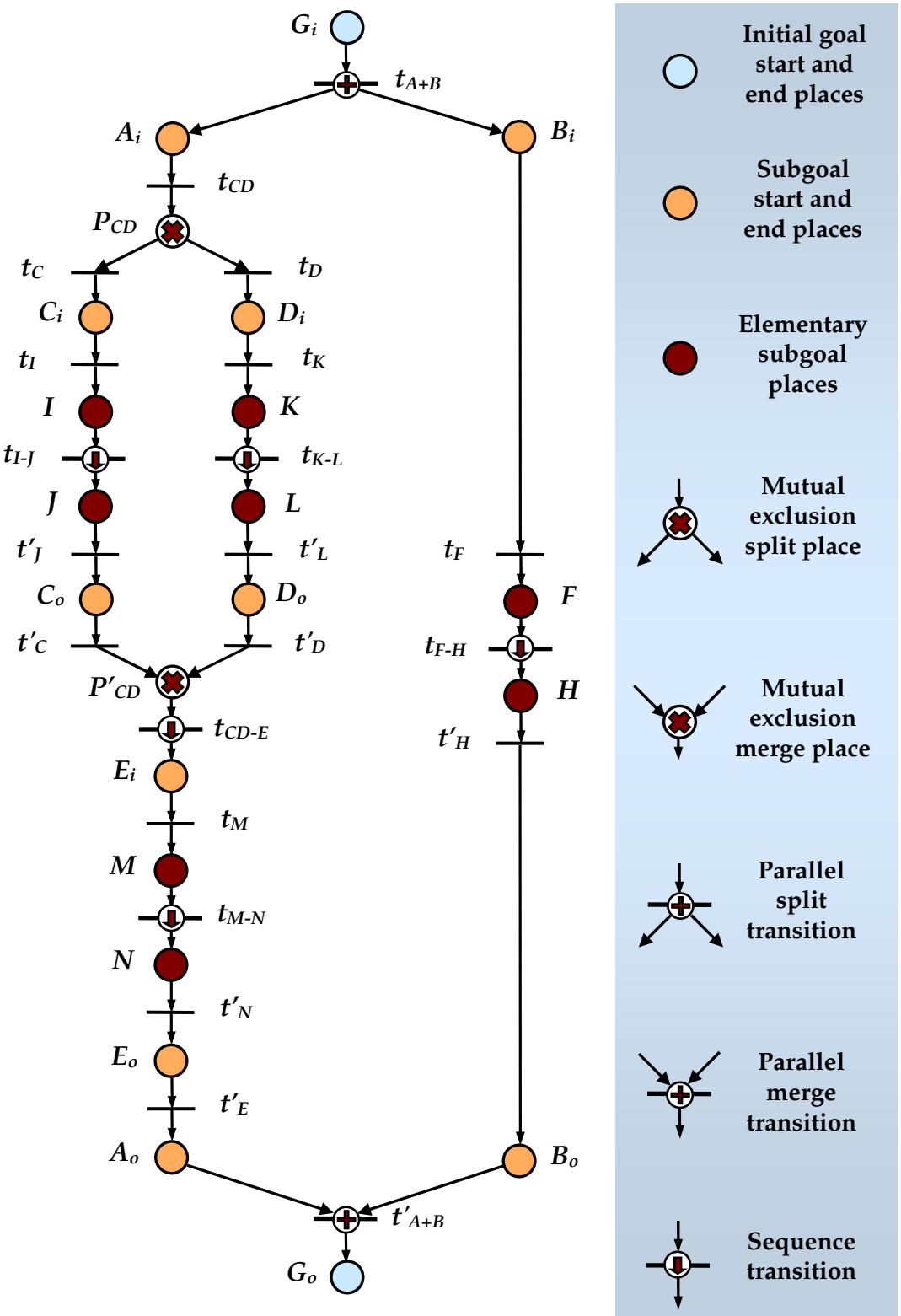

FIGURE 88 Generic service net of the goal “make trip”

In a service net transitions encode actions. All transitions can be classified into service actions, control actions and auxiliary transitions. Service actions are the meaningful functionality performed by services. Control actions are transitions that belong to composition constructs and encode fulfillment of certain control functions, such as control flow splitting and merging. Auxiliary transitions do not play any significant role. They are mostly used to build a vivid graphical representation of service nets.

In a generic service net no service actions can be present, since these actions belong to services, which are absent in a generic service net. So, a generic service net may only contain control actions and auxiliary transitions. The examples of control actions from the generic net of Figure 4.17 are transitions t'_{A+B}, t_D and $t_{K\text{-}L}$. The examples of auxiliary transitions are t_{CD} and t'_H. Auxiliary transitions are usually incident to non-elementary subgoal places. Since non-elementary subgoal places can be usually removed from the service net without serious consequences, the auxiliary transitions can also be removed together with them.

Speaking of the physical sense of the built Petri net model, its named places corresponding to the elementary subgoals should be treated as slots for real services. Basically there must be services available in the environment, which actually perform tasks that achieve these elementary goals. In this way we reserve the places associated with the elementary goals as empty services. These empty services are going to be filled or substituted with some real services or collections of services at the stage of service delegation.

However, there might be available services, which perform more complicated tasks than those achieving elementary goals. It is quite probable that there exist services achieving non-elementary subgoals or even the main goal. Of course, the possibility to utilize such services should also be provided when creating composition models. To achieve this, a Petri net model structure should be extended with additional places and transitions associated with non-elementary subgoals. It can be easily seen that in the previous example of the generic service net structure for the goal “make trip” only places associated with the elementary goals were present (except for some auxiliary places). Now, let us extend the structure of the Petri net shown in Figure 88.

Figure 89 presents a generic service net for the goal “make trip”, which includes empty services for some non-elementary goals. To be more specific, such empty services can be included into the service net only for potentially elementary subgoals, i.e., subgoals which are non-elementary but have valid service counterparts. For the sake of simplicity and pictorial clarity we added empty services into the service net shown in Figure 89 only for the subgoals A and B, assuming that these subgoals were classified as potentially elementary during the goal decomposition process.

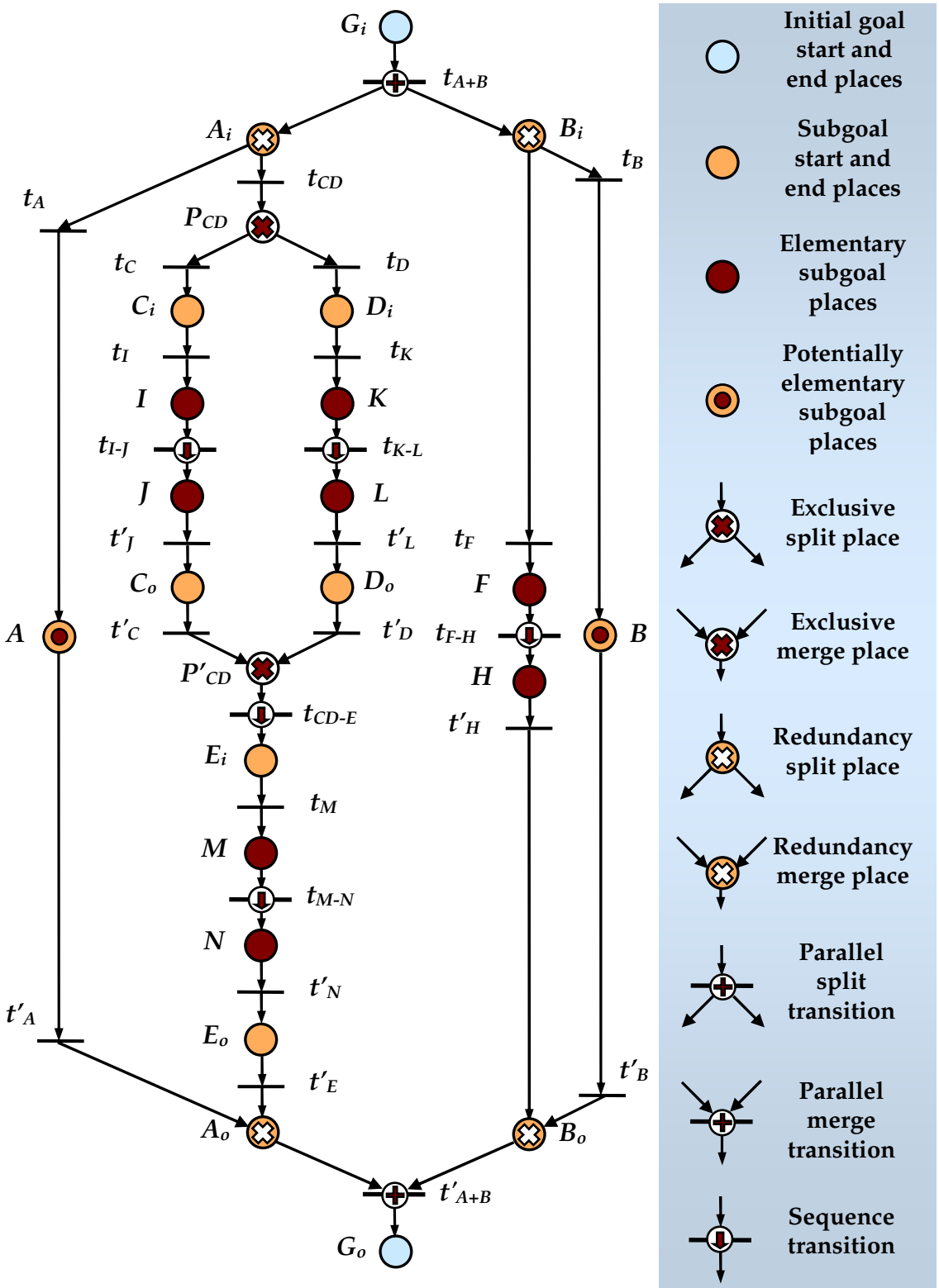

FIGURE 89 Generic service net for the goal "make trip" with empty services for potentially elementary subgoals A and B

It should be noted that in order to add empty services for potentially elementary subgoals into the generic service net, the starting and ending places of the corresponding subgoals (A_i, B_i, and A_o, B_o respectively for the service net of Figure 89) must be modified into mutual exclusion constructs called *redundancy splits* for starting places and *redundancy merges* for ending places. These mutual exclusion constructs operate exactly in the same way as ordinary mutual exclusion. However, to their composition-related role we will return later.

Summarizing this section we say that goal-driven control flow construction for service composition takes a valid goal decomposition model as input and produces a corresponding Petri net control flow structure, which we call a generic service net. Generic service net, such as one illustrated in Figure 89, reflects the order of the corresponding goal achievement. It does not contain, however, any real services, instead it reserves slots for them in the form of empty services. Generic service net is constructed to be as flexible as possible and to allow non-deterministic execution by exhibiting redundancy with alternative execution sequences. This redundancy is achieved by including, into the service net, alternative execution paths for all potentially elementary subgoals. So, the execution plan for each such subgoal has two exclusive options: one implies execution of composition of multiple services, which correspond to the subgoals into which the given subgoal was previously decomposed; the other refers to invocation of a single component service which is capable of achieving this subgoal on its own. After such a generic service net is built, the composition planning process is in principle over. Next stages are mainly devoted to manipulations with component services to be put into the generated composition plan.

4.3.5 Component Service Discovery

Service discovery refers to a process of finding services. What is more, services to be discovered are usually somehow pre-specified within a request for discovery, i.e., user basically wants to find appropriate services with respect to certain requirements. These requirements are generally applied on services' capabilities and associated data formats. Finding a proper service, which conforms to user demands, often is a non-trivial task due to possibly huge number of service repositories, different service description standards and ontology languages. However, efficient and, most importantly, automated service discovery cannot be imagined without paying attention to all these varying factors.

Service discovery process is tightly coupled with the process of service matchmaking, since all unsuitable services have to be discarded once discovered. Service matchmaking is the procedure that checks discovered services' appropriateness for the goals specified by the user. In the context of service composition, the task of the service matchmaking procedure is to select proper

services for assembling them into the given composition plan. We will consider this procedure in detail in the next section.

What is quite important about service discovery in the context of automated service composition is that the service discovery process has to be completely automatic and it must provide the service matchmaking process with the services that can be actually matched into the given composition plan.

We do not intend to focus on the automation requirement since service discovery is a widely recognized research domain that can offer a variety of solutions for service discovery automation. We only note that the crucial point here is the availability of service repositories and the universal standard for service description and discovery. Universal Description, Discovery and Integration (UDDI) [27] is currently the prevailing common standard on service descriptions and discovery, and provides the base for discovery process automation. However, to promote full-fledged service discovery automation, the progress of semantic service discovery mechanisms is necessary.

The second requirement we put on the service discovery mechanism is much more interesting. Even though the service discovery mechanism can discover services automatically, it should find services which can be further matched to the user requirements and goals. Unfortunately, this is a much more problematic point. In order to perform service matchmaking a service must be described in the way that it is comprehensible what kind of functionality this service performs, what are the preconditions for its use, what are the effects it produces as the result, and what is the meaning of its input and output data. Unfortunately, the current service description standards WSDL [26] and UDDI [27] lack needed formal semantics to answer these questions and to provide those "answers" in a machine understandable format. While WSDL is intended for syntactic specification of service invocation, e.g., for description of input and output message formats, UDDI additionally provides a possibility for description of service capabilities in natural language. Since such textual descriptions are not semantically annotated, this UDDI's feature is of little help for automated discovery and matchmaking frameworks.

In order to allow for full automation of service matchmaking procedure, services have to be semantically annotated using some formal machine-understandable descriptions. Semantic service specifications can be built using specific ontology standards and appropriate domain ontologies. Two major semantic service description standards are independently being developed by W3C consortium and European Semantic Systems Initiative (ESSI). They rely on the use of semantic description languages OWL [70] and WSML [20] respectively for building semantic service descriptions, and service specification ontology languages OWL-S [66] and WSMO [95] respectively for creating service vocabularies for semantic Web services. Semantic service specification languages allow describing services' capabilities, inputs, outputs, preconditions and effects in

the form of logical expressions, the truthfulness of which is to be verified during service matchmaking. Service specification ontologies, in their turn, describe the concepts which are used for building semantic service descriptions. Thus, an automated service matchmaking procedure takes a semantic service description written in OWL or WSML, understands its content using the corresponding OWL-S or WSMO service ontology and appropriate domain ontology to which the investigated service is related, and finally matches certain statements within the provided service description to the given set of requirements to verify whether the service is appropriate for achieving the user's goals or not.

We assume that our Web service composition framework utilizes precisely this semantic approach for service discovery and matchmaking. We do not have any strong requirements on supported service description formats and service specification ontologies, since designing the composition reasoner does not belong to the contributions of this thesis. We describe the service composition planning process in a rather generic manner, focusing only on the aspects which we contribute to.

As we already mentioned while describing the goal decomposition procedure, service discovery and partial service matchmaking procedures operate in two phases within our service composition framework. First service discovery and partial matchmaking are performed during the goal decomposition process in order to determine elementary subgoals and in certain cases even force specific subgoal decomposition. Services, which are discovered and partially matched at this stage as counterparts to certain subgoals, are not abandoned. They are memorized as potentially valid component services in order to perform availability check on them later at the second phase of service discovery. By partial matchmaking we mean that, during goal decomposition, services are basically tested with respect to their capability to achieve the given goal. If a certain service, discovered at the goal decomposition stage, performs the task that completes the given goal, then that service is said to be partially matched as a candidate component service.

The second phase, when service discovery and matchmaking is performed, is fully devoted to discovery and matchmaking of all available component services which can be put into the recently constructed composition plan as various execution alternatives. Service discovery process is accomplished the same way as it is done during goal decomposition. Service matchmaking is, however, done with some differences, which we specifically describe in the following section.

4.3.6 Service Matchmaking

As we already stated in the previous section, service matchmaking is the process of verifying the suitability of a service to achieve given goals and to fulfill given requirements.

Automated service matchmaking procedure, which is among our goals, can operate only with well-defined semantic service descriptions written in a semantic service specification language, such as OWL or WSML. While not delving deeply into concrete formats of semantic service descriptions, we should nevertheless emphasize the main issues involved. A semantic service description should at least cover the following service-related aspects:

- *Service capabilities*: the essence of the tasks the service is able and aims to perform;
- *Service preconditions*: the set of prerequisites for the service invocation and flawless execution;
- *Service effects*: the set of world-altering events produced as a consequence of the service execution;
- *Service inputs*: the meaning of the data structures the service takes as its input;
- *Service outputs*: the meaning of the data structures the service produces as its output as a result of its execution.

Service matchmaking is the main reasoning procedure, and the task of the reasoner is to analyze semantic service descriptions of the available services and to match them to specific places within the generic service composition plan. More specifically, the reasoner considers every subgoal included into the composition plan and finds a set of matching component services which are capable of achieving this particular subgoal. To find such a match, the reasoner analyzes service capabilities of the available services and infers whether certain service matches a given goal or not. We can call a service matched in this way *semantically eligible*. For certain services this type of matchmaking is made already at the goal decomposition stage.

However, finding matches to a goal is not the only matchmaking procedure the reasoner performs. It also has to verify that a service matched to a goal is eligible for being included into a specific place of the current composition plan. We can say that a service is *causally eligible* if all its preconditions are satisfied. Service preconditions usually refer to a certain state of the part of the world, or a certain state of affairs which is somehow affected by the service's operation or is related to the service in some other way. Usually this state of affairs can only be changed by the service itself and by other services included into its service composition. So, the basic rule for a proper sequential composition of two services is that the preceding service has to produce an effect that produces a state of affairs which is required by the precondition to the following service.

Services' preconditions and effects are encoded by logical expressions, which is a convenient form to reason over. Some of the mature reasoning approaches for

semantic service composition are based on the situation calculus formalism, which is a powerful and concise tool for manipulating service effects and preconditions.

Functionally eligible component service can be defined as a service which inputs can be directly matched to the outputs of the preceding service within the composition plan. This type of matchmaking is quite straightforward. Each service is supposed to get some data prior to its operation phase. The data a service requires are precisely defined in terms of format and meaning. So, prior to including a candidate service into the composition plan the reasoner must verify that the inputs required by this service accurately match the corresponding outputs of the services, which precede this service within the associated data flow. The necessary condition for including a candidate service into the plan is that its inputs match in terms of their meaning. The sufficient condition is that they match in terms of their format. Nevertheless, if the meaning check is passed successfully, but the format check fails, it is possible to incorporate an appropriate data translator facility into the composition. Such modification is, however, beyond the scope of our design, since we do not model data flows between services.

As soon as all matchmaking procedures complete and all candidate services are determined, the reasoner can extend the composition plan with the places for alternative candidate services if such services remain after the matchmaking phase. For example, if the reasoner determined two different candidate services achieving a subgoal A within the composition plan, it subjects the corresponding place A of the service net to the operator of mutual exclusion that has two alternative places A_1 and A_2. The choice between the alternatives A_1 and A_2 is supposed to be made during execution either randomly, or by a separate contextual control, which we'll consider later, or on the basis of non-functional attributes of the associated component services, such as their cost, time of execution, etc. We will discuss non-functional attributes of component services and their influence on exclusion operators in the forthcoming sections. The choice between three and more alternative component services can be made by the extended version of the mutual exclusion operator, which allows more than two alternatives to be added. The corresponding extension to the generic composition plan is illustrated in Figure 90. Within the service net of Figure 90 two empty services are reserved for a component service achieving subgoal A, and three empty services for services achieving subgoal H.

As soon as the service matchmaking phase is over, the places within the composition plan are reserved for all selected services and the services can be put into the plan. However, a separate service delegation procedure is dedicated for putting component services into the plan.

4.3.7 Context for Composition Refinement

Now it is clear how a generic structure of a composite service net model is built. The next step is to build a particular model of a composite service. A particular composition model results from the generic composition model in two steps: through another reduction of the generic model and by putting component services into the model. Reduction is achieved by cutting off some alternative paths (corresponding to alternative subgoals and/or alternative component services achieving the same subgoal) of token propagation through the service net in favor of other paths.

For example, in the context of our traveling example we may have, say, two options of how to book the whole trip. One option is to utilize a separate service to book a flight to London and a separate service to book a taxi from the airport to a hotel, while the other option is to use the service which accomplishes both of these tasks. The choice is often not easy. We also have to consider some non-functional characteristics of both options, such as quality of service, usage cost, reliability and so on. Making the choice between alternatives on the basis of their QoS parameters is the simplest possible scenario. To make such a choice more relevant, we should also consider some specific services' requirements, which impose additional preconditions on services' usage. How these preconditions differ from normal preconditions checked during the service matchmaking phase? Normal preconditions, as we mentioned above, usually deal with some implicit state of the service execution process or with the current state of relevant affairs. Additional preconditions we are currently discussing are of a more general type. They address certain logical facts which may refer to any place of the entire observable world. These facts may address conditions from a physical world environment as well as from a computing environment and from the user. Needless to say that such facts may cover a huge variety of conditions, which are not always easy to check. These conditions can be referred to as context, and the corresponding facts as *contextual preconditions*.

Simply put, to perform a grounded choice between alternative services, we have to check the context of the available services, verify their contextual preconditions and make a final choice. So, in short a generic Petri net model of a composite service should be reduced, and the reduction should be made based on the context.

The main question here is how to acquire the context we need. Component services' context is the crucial thing for ensuring the composite service's operability. It is likely that if a certain component service fails to operate under certain conditions, the whole composite service will also fail. So, it is hard to overestimate the significance of the information describing which conditions are favorable for each service and which are not.

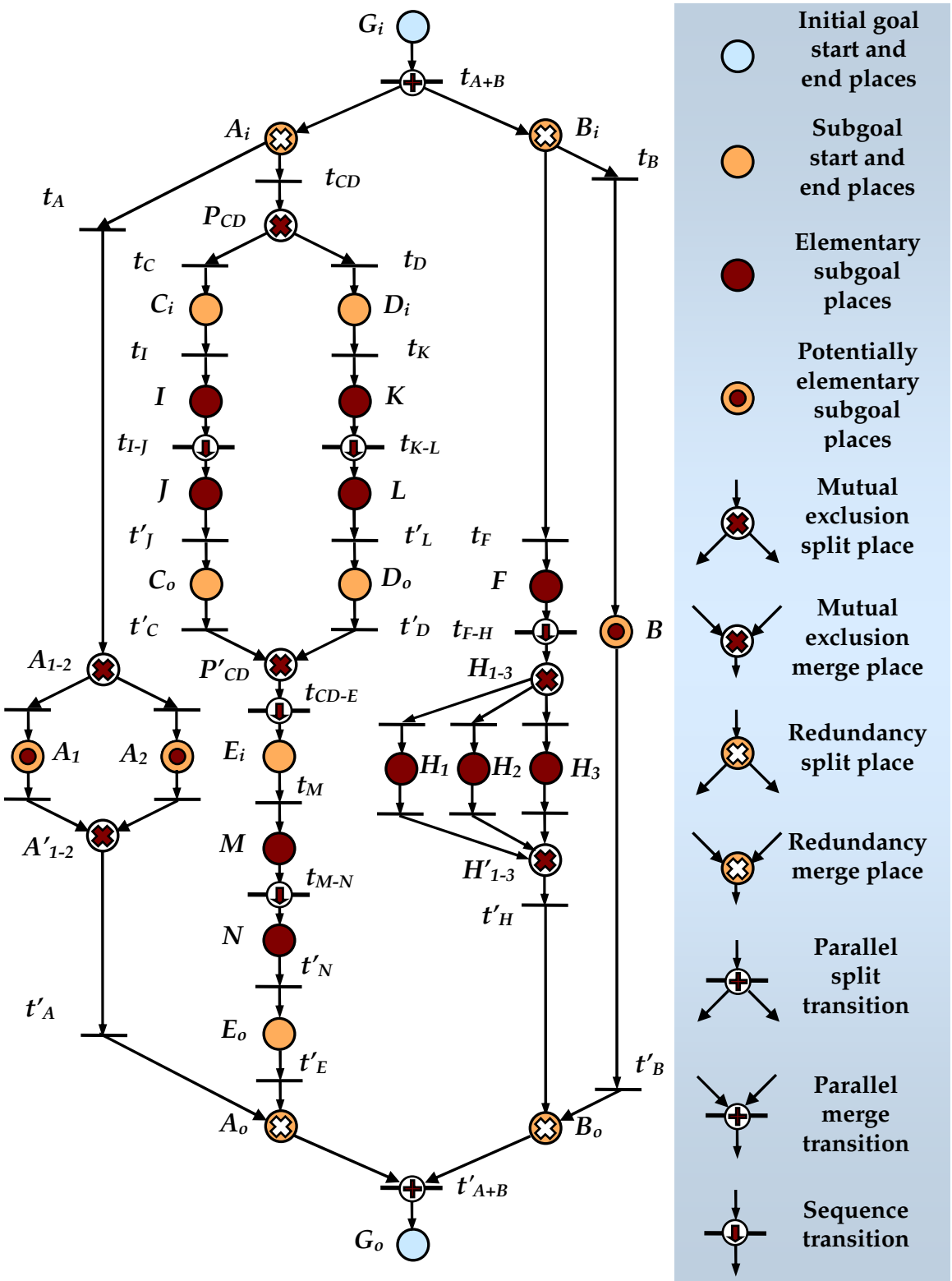

FIGURE 90 Generic service net with empty services for alternative component services achieving subgoals *A* and *H*

Unfortunately, there is no common source or way for describing and acquiring service context. Web services do not have a recognized standard for describing service context. Surely, there are numerous frameworks for description and attachment of metadata to Web services and for building service descriptions. However, this type of information can be hardly classified as context.

Metadata is used to describe a service itself, while context is the knowledge about the environment the service is currently immersed in. So, besides the information of what the service does and how, we also need to know when, where, and under what conditions this happens. Having such information, we should be able to check the current conditions and decide which service is best suited for the present moment.

One of the possible solutions is to extend the Web service description mechanism to include specific contextual precondition descriptions. The only requirement is that it has to be a semantic description, so that the reasoning engine will be capable of processing it. To become semantic, a description should be regulated by ontologies, i.e., all the data objects declared in the description have to be the instances of certain ontologies. Basically, existing service descriptions are to be extended with a special construct, which can be called "true in context", that is a container of contextual statements and/or other contextual containers. Each such statement binds a value or a range of values to a certain context variable or a combination of variables, thus describing the set of conditions under which the service can be utilized. A detailed description of such an extension to service descriptions can be found in [54].

Another interesting issue concerns the stage of service lifecycle, at which such context-based reduction is to be performed. Alternatives can be cut out either on the stage of composition planning or on the stage of composite service execution. The latter option implies two things. First, the mechanism of composite service building should allow real-time reduction or reconfiguration of the service structure. Second, there should be available means for real-time acquiring, processing and communicating of context to composite services.

As we may recall from Section 2.1, context is generally a highly dynamic matter, i.e., it may change faster or more frequently than a single service execution period. Thus, most of the available contextual preconditions of component services have to be checked in real-time fashion, just before the actual start of the corresponding component services' execution. Those contexts which are rather stable with respect to the duration of composition planning and service execution phases can be checked during the planning stages. The principles of such contextual preconditions verification are nearly the same as for the normal preconditions.

4.3.8 Service Delegation

As soon as a generic Petri net structure of a composite service is built, service delegation procedure can be started. Service delegation is basically a process of building a particular composite service plan from the given generic composite service plan. The major distinction between a particular and a generic service plan is that the service net of the particular plan includes real component services instead of empty services present within the generic service net. So, the main goal of the service delegation procedure is filling a generic service net with real component services.

The composition reasoner already knows which component services are to be used prior to starting service delegation. Eligible component services are discovered during service discovery stage and proven for the current plan during service matchmaking. So, the reasoner should simply take these prepared services and rebuild the generic Petri net structure of the composition via substituting empty component services with real candidate Web services using the refinement operator. This process corresponds to the modeling from scratch paradigm described in the Chapter 3.

Figure 91 shows how the generic Petri net structure of the composite service achieving the goal "make trip" looks like after such a substitution. It can be easily seen that in comparison to the model depicted in Figure 90 this composition model does not contain any elementary or potentially elementary subgoal places. All of these places are considered empty services and substituted with real services during service delegation. So, empty service places A_1, A_2, B, F, H_1, H_2, H_3, I, J, K, L, M, N from the net of Figure 90 are substituted with real services Sa_1, Sa_2, Sb, Sf, Sh_1, Sh_2, Sh_3, Si, Sj, Sk, Sl, Sm and Sn respectively. In this way we obtain a purely composite Web service achieving the goal "make trip". Now its service net model can be in principle subjected to examination and further implemented as a real composite service.

However, finalizing the model at its current state is in many cases hasty and not reasonable, because the model can be still too large in terms of the number of alternative component services. Including alternatives into the model should not be subordinated to the criterion of maximizing the number of alternative execution paths. Conversely, we have to strive to place, into the model, as few alternatives as possible, but alternatives that are really able to improve the quality of the service. Whenever we insert an uncontrolled exclusion construct into our service we increase the service's complexity significantly, while gaining almost nothing in quality. Purely non-deterministic exclusion is only good when speaking about fault tolerance: if one alternative suddenly fails or becomes unavailable, the other can be easily utilized then. But for enhancing the service functionality, uncontrolled exclusion is almost useless. So, we should reduce the number of alternative services and abandon uncontrolled exclusions as far as possible to

minimize the complexity of composite services without degradation of their quality. To achieve the first objective of discarding all retractable alternative services from the plan we have to utilize contextual and non-functional service properties where and if it is at all possible. Context can be utilized as a basis of service net reduction when corresponding alternative services have contextual preconditions within their corresponding semantic service descriptions. If it can be guaranteed that the context required by a certain contextual precondition is sufficiently stable to not change during the period of time until the anticipated completion of the service, then such context can be checked already at the service delegation stage and the corresponding precondition can be verified. Thus, if some contextual preconditions of an alternative component service have been proven to fail, then this particular service is simply discarded from the composition plan.

For example, we can imagine a composition plan containing a flight booking component service. This service has a contextual precondition requiring that there should be a favorable flying weather for the departure of the user. The reasoner may invoke a weather forecast service to find out that the current weather is non-flying and is not likely to change within the forthcoming 24 hours. So, the user won't be able to depart later that day using air plane. The reasoner infers that the precondition is failed, and discards the flight booking service from the plan. However, if the weather forecast shows that the weather might change in the following several hours, it cannot be guaranteed that the precondition has definitely failed and its verification has to be postponed to the service execution phase.

The described type of contextual reduction can be successfully applied to normal exclusion splits, which embrace actual services. Unfortunately, it is not that easy to apply such retraction procedure to reduce redundancy splits. Redundancy exclusion splits basically correspond to the choice among different alternatives of the same non-elementary subgoal achievement. Non-elementary subgoals are usually complex and can be potentially elementary in the best scenario. This implies that redundancy splits generally embrace not the single services but sophisticated sequences of services. The aggregate context of such sequences is extremely difficult and cumbersome to infer and verify. Therefore, contextual reduction is almost completely non-applicable to redundancy splits unless specific goal ontologies, which particularly describe contextual preconditions of their corresponding subgoals, are established.

Another important observation, which we already mentioned before, is that context is rarely that steady to verify contextual preconditions prior to the service execution phase. Therefore, if none of service's contextual preconditions have failed so far at the stage of composition planning and if there remain contextual preconditions which cannot be sensibly checked at the planning phase, the service remains in the model and its preconditions will be checked during the execution phase.

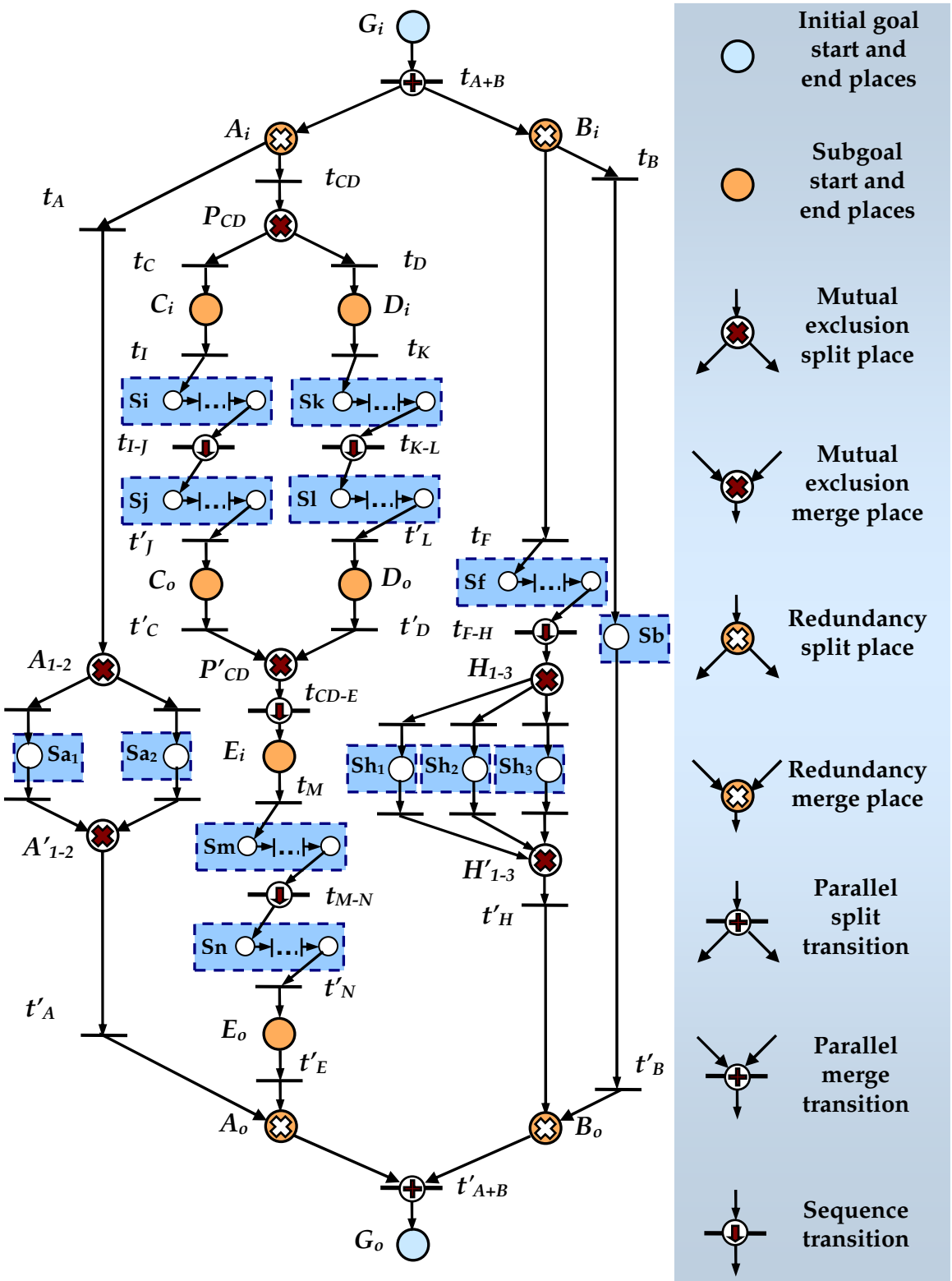

FIGURE 91 The "make trip" service net filled with delegated component services

In case when component services and subgoals are not supplied with contextual preconditions or when their preconditions check has not produced any grounds for performing reduction, they can still be retracted on the base of their non-functional properties. By non-functional properties we particularly mean Quality-of-Service (QoS) characteristics and requirements of services, such as usage cost, average execution time, resource requirements, etc. Two services or service sequences embraced by a single exclusion split can always be compared according to their non-functional characteristics, and the inferior service can be retracted in favor of the superior one. It should be noted that it is not always possible to unambiguously decide which service is better and which is worse with respect to their QoS characteristics. So, the non-functional reduction mechanism has to be provided with some criteria for grounded decision making. These criteria can be either set to default criteria supplied by the reasoner, or provided by the user explicitly within the composition request, or extracted automatically from his/her client. The latter option also reflects the context-aware paradigm and implements a piece of personalization functionality within our service composition framework.

Though non-functional properties of all component services can be easily checked, it is not always reasonable during the service delegation stage. First of all, it should be emphasized that non-functional properties should be accepted as the primal decision factor only after all other factors have been considered. The only exception to this rule is service availability. But this factor is basically checked during the service discovery stage. So, if contextual preconditions of a service have not been checked during service delegation, non-functional reduction cannot be started and should be postponed to the execution phase.

Similarly to contextual preconditions, non-functional properties can not always be sensibly checked prior to execution because such factors as available resource levels are simply not known in advance. However, there are non-functional factors such as service usage cost, which is nearly constant, so the reasoner can use them at any suitable time to make decisions. Nevertheless, if a service features some important non-functional properties that cannot be sensibly tested during service delegation, non-functional reduction should be postponed to the execution phase.

Figure 2 illustrates results of both contextual and non-functional reduction performed on the generated “make trip” service net. Service Sh_2 is the payment service for hotel booking, and performs payments using credit cards. It was retracted after the reasoner inferred such contextual fact that the user does not own a credit card, which is obviously a steady context. Service *Sb*, which books hotel accommodation, was retracted by a non-functional reduction due to its unacceptably high cost for the user.

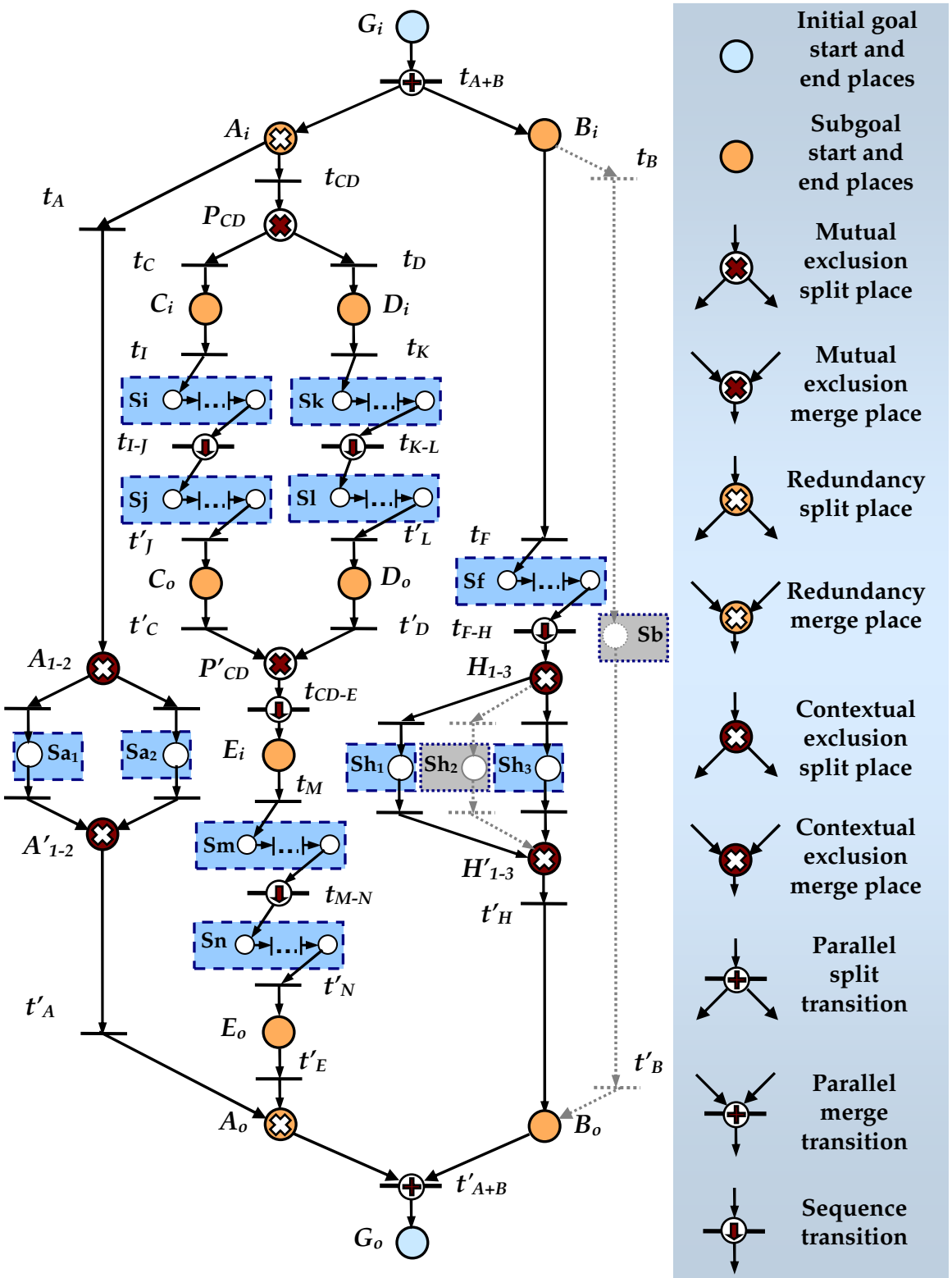

FIGURE 92 The final service net for the “make trip” service composition

It can also be seen that the redundancy split and merge places B_i and B_o after the reduction of the service S_b become simple places, since no alternatives remain within their bounds. The retracted services are removed from the model with all the incident arcs and transitions (shown in grey in Figure 92) within the bounds of the corresponding split-merge pairs.

As soon as all possible reductions are made the remaining exclusion splits become *controlled* exclusion splits and can be classified into two distinct categories. *Contextual* splits embrace alternatives, the choice among which is to be made during the execution phase with respect to context. The rest of the splits are *non-functional* splits, which embrace alternative services lacking contextual preconditions or whose contextual preconditions have already been verified. For these splits the choice among alternatives will also be made during service execution, but with respect to services' non-functional properties.

The reason why we completely got rid of uncontrolled exclusions is that the controlled exclusions have a role to play as the main service quality and flexibility assurance factors. They help ensure that no matter which service choice is made during the execution of the composite service, it is not made blindly but deliberately and to achieve the best possible results from the user's perspective.

Now the construction of the composite service net is over. However, the planning phase of the context-aware Web service composition framework's operation is not yet finished. If controlled exclusion splits can be located within the final service net structure, the service net alone is not enough to finalize the composite context-aware Web service. In this case an additional Petri net model should be constructed. We call this model a *control net* or control layer. This additional Petri net model is a completely independent net structure that can be operated autonomously and independently from the corresponding service net and is supposed to establish external control over controlled exclusion splits of the associated service net. We describe the control net construction procedure with a higher level of details in the following section.

4.3.9 Contextual Control Functionality Construction

As we already stated above, contextual control net construction is the last stage of the context-aware service composition planning process. However, it is an optional stage because the control net construction procedure is instantiated only if the newly generated composite service net contains some controlled exclusion constructs. The objective of the control net construction procedure is to provide these service net exclusion constructs with appropriate real-time control.

A *control net* is a Petri net structure, which generally consists of an appropriate service net structure and of some specific constructs exclusively utilized within control nets.

Here we present neither the principles of the design nor samples of control net structures. We feel that for reasons of clarity and ease of understanding it would be more logical and convenient to investigate these aspects of control net construction under the service composition reconfiguration section. Here we only describe the actual process of contextual control functionality construction without delving into the mentioned modeling issues.

The contextual control functionality construction process can be roughly split into the following procedures:

1. Context provider service net construction (optional).
2. Control facilities attachment.
3. Control net linkage to service net.
4. Composite service finalization.

Procedure 1 is in fact the same planning procedure which we have described so far in Section 4.3 with some distinctions. The aim of context provider service net planning is to provide a context provider service which checks certain required context in a real-time fashion. By context provider service we understand an information Web Service, the main function of which is to present necessary information to a customer with respect to a given request. Context provider service can be either atomic or composite. The customer of such context provider service is the composition reasoner.

The procedure of context provider service net construction is optional in the sense that it can be omitted if an appropriate ready-made context provider service can be discovered and used to acquire necessary context. Otherwise, such a service has to be composed in exactly the same way the user requested service in question is composed. So, the service composition planning procedure has to recursively call itself to build a composition inside the composition. In Figure 78, which describes the overall context-aware Web service composition process, the context provider service construction procedure corresponds to the steps 8-16. We do not intend to describe these steps again, since they are almost the same as the steps 1-7 for the main service planning. There are only two differences. One of the differences is that the composition request for the context provider service is generated automatically by the reasoner. Another difference is more sensible: for the context provider composite service, only its service net is constructed but no additional control net is attached to it.

As soon as the context provider service is ready, the second procedure is started. It attaches special control constructs, such as, for example, semantic transitions, which will be further described in the next section, to the given context provider service in a way that will allow real-time verification of a given contextual and/or non-functional precondition of a given component service from the associated service net. The way in which this procedure links control constructs

will also be described in the next section. Thus this procedure corresponds to the step 17 in Figure 78.

Procedures 1 and 2 are repeated as many times as needed to construct contextual control facilities for all the component services' contextual preconditions to be checked during the composite service execution.

As soon as all the necessary contextual control facilities have been constructed, procedure 3 starts to link these facilities to the corresponding exclusion splits from the associated service net using Meta Petri nets organization principles. This procedure corresponds to the steps 18 and 19 in Figure 78.

When inter-layer linkage is done, the user requested composite service can be examined, verified and finalized. This corresponds to the step 20 in Figure 78.

Thus, we have described the process of goal-driven context-aware construction of composite Web services. Our planning framework flexibly constructs composite service with respect to the user goal and based on the current context. So, we have showed how to automatically construct composite services in a context-aware fashion. However, it is still to be shown how to construct context-aware composite Web services and reconfigure them at run-time. We discuss this crucial issue in its related aspects in the following section devoted to context-aware dynamic real-time reconfiguration of composite service structures.

4.4 Context-aware Dynamic Reconfiguration of Composite Web Services

As we already described in Section 4.2.3 reconfiguration takes place at the run-time or execution phase of the context-aware Web Service's lifecycle. In Section 4.2.3 we gave an explanation of the basic principle of context-aware Web service's reconfiguration. Here we concentrate on the representation of context-aware Web services in terms of Petri nets and show how reconfiguration is made using Petri nets' operational semantics.

First we show how context-aware Web services are modeled using Petri nets. Then we rigorously investigate the process of modeling contextual control functionality to be included into Web Services. After that we describe how exactly contextual control is embedded into a service. Finally, we demonstrate the actual process of a context-aware Web service execution, emphasizing operations that actually reconfigure the service.

4.4.1 Context-aware Web Services as Petri nets

Similarly to the Web service definition given in Section 3.2 we give the following definition for a context-aware Web service.

Definition 4.10. *Context-aware Web service* is a tuple *S*=(*N, D, L, URL, CS, CP, CSN*) where:

- *N* is the name of the service, used as its unique identifier;
- *D* is the service description;
- *L* is the location of the service;
- *URL* is the invocation of the service;
- *CS* is a set of component services constituting the service. If *CS*={*N*}then *S* is an atomic service. Otherwise it is a composite service;
- *CP* is a set of context provider services included into the service;
- $CSN = (P_S, P_C, T_S, T_c, F_S, F_C, IL)$ is the context service net modeling the context-aware Web service *S*.

From this definition we can see that the main difference between ordinary services and context-aware services is in their service nets. A context-aware Web service is modeled by a service net, which is called context service net, and comprises two distinct net layers tied together by Meta Petri nets principle. We can also see that a composite context-aware Web service has two component services' sets: one includes normal component services and the other component context providers.

A context service net of a context-aware Web service consists of two separate Petri net structures: service net and control net. These two nets are placed on two hierarchical layers to be further linked by a special hierarchical control linkage.

4.4.1.1 Service Layer

A Meta Petri net *CSN* modeling a context-aware Web service consists of at most two layers, as we have already said. The lower layer is the service layer and contains the service net *SN* of a service. We do not intend to pay much attention to service layer here, since we devoted the whole Chapter 3 for describing service net modeling principles.

The only important thing to note about the service layer is that a service net residing on it has to contain at least one controlled exclusion split construct, which will be further linked to the control layer. Otherwise, a Meta Petri net of a service cannot be built, so a context-aware Web service cannot be modeled in consequence. Also, the service layer cannot contain more than one service net, because it is supposed to model a single service, which is described by a single service net.

Another interesting observation is that the service net should be built in such a way that it can be executed even without the presence of any control net, i.e., despite any controlled splits present in the service net, its output transition is live (see Section 3.4).

4.4.1.2 Control Layer

Control layer is the upper layer of the Meta Petri net *CSN* modeling a context-aware service. Control layer is supposed to provide a specific Petri net functionality which performs control over controlled exclusion splits of the corresponding service layer. A control layer may contain one or more control nets. The number of control nets residing on the control layer is basically defined by the number of the controlled exclusion splits on the corresponding service layer.

Definition 4.11. A *control net* is a tuple $CN = (P, T, F, i, o)$ where:

- P is a finite set of control places;
- T is a finite set of control transitions, such that $P \cap T = \varnothing$;
- F is a flow relation; i.e., a set of directed arcs $F \subseteq (P \times T) \cup (T \times P)$;
- i is the input place such that $\forall f \in F, f \notin I(i)$, i.e. $I(i) = \varnothing$ (where $I(i)$ is the input function of the place i);
- o is the output place such that $\forall f \in F, f \notin O(o)$, i.e., $O(o) = \varnothing$ (where $O(o)$ is the output function of the place o);

It is obvious that such a definition of a control net is not any different from the definition of a service net (see Section 3.2). Indeed, from the formal viewpoint a control net is an ordinary Petri net in a standard form just the way a service net is. However, we will extend this type of Petri net structure with additional semantic constructs in Section 4.4.2.

It is important to note that multiple control nets belonging to the control layer are generally not connected to each other. So, each control net is a completely autonomous structural unit, which is executed independently of other similar units.

Control nets are basically partially composite services. Each control net in the majority of cases contains at least one component context provider service, which performs real-time context check. Since each control net is associated with one controlled exclusion split of the service net, it may contain as many component context provider services as there are unverified contextual preconditions over all component services subjected to the corresponding controlled exclusion operator on the service layer.

By context provider services we understand such information Web Services which check either certain contextual variables or non-functional parameters. A context provider service can belong to composite services as well. As we already mentioned, context provider composite services can be built by our framework in the very same way the normal composite services are built. So, composite context provider services have their own service net that links an arbitrarily large number of component context provider services. These service nets become a valid part of a control net after context provider service composition is completed. However, composite context provider services generated by our framework cannot themselves be context-aware. This is an artificial limitation we inflict deliberately

because its absence would potentially produce an infinitely long sequence of nested meta-layers within the service model, and would result in a composition solution neither scalable nor efficient. Thus, only the externally provided atomic context provider services can be context-aware.

The control layer functionality can be split into several functional procedures, which are permanently present within any control net. These procedures are:

- *Service Validity Check*. This procedure checks whether a given component service can be applied to a given composition with respect to current context. This procedure can be logically decomposed into context acquisition and contextual precondition verification sub-procedures, the number of which corresponds to the number of the associated contextual preconditions of the service to be verified. We will further denote pairs of context acquisition and contextual precondition verification procedures simply as precondition check.
- *Conflict Resolution* procedure aims at deciding which service among all the alternative services to apply on the service layer. This procedure either chooses one service based on the result of services' validation if there are no conflicts, or passes the decision making responsibility to one of the other procedures depending on what kind of a conflict it encounters.
- *Non-functional Service Selection*. This procedure is applied whenever multiple services are validated with respect to context, i.e., the conflict resolution procedure encounters a multiplicity conflict. The aim of non-functional service selection is to choose the best service among all validated services based on their non-functional properties, i.e., QoS characteristics.
- *User Aware Adaptation* is a procedure of inferring a corrective action and providing it to the user. Such corrective action is a proposition to the user to change context, through a physical action, in such way that some of the services can be validated in it. This procedure is enacted whenever conflict resolution procedure receives no validated services from the service validity check.

The structure of the contextual control functionality in terms of the described procedures can be shown in the form of a flowchart as presented in Figure 93.

As it can be seen from that figure each contextual control function and its corresponding control net contains as many service validation procedures as there are services from the service layer subjected to the associated controlled exclusion operator. Similarly, each service validation procedure comprises as many precondition checks as there are contextual preconditions of the associated service to be verified.

The structure of the Figure 93 can be easily mapped into an adequate control net, which we will show in forthcoming section. What should be emphasized now is that this general structure corresponds to exactly one control net. The control layer may contain a number of such control nets, each of which corresponds to a separate contextual control function operating over a single controlled exclusion operator from the associated service net.

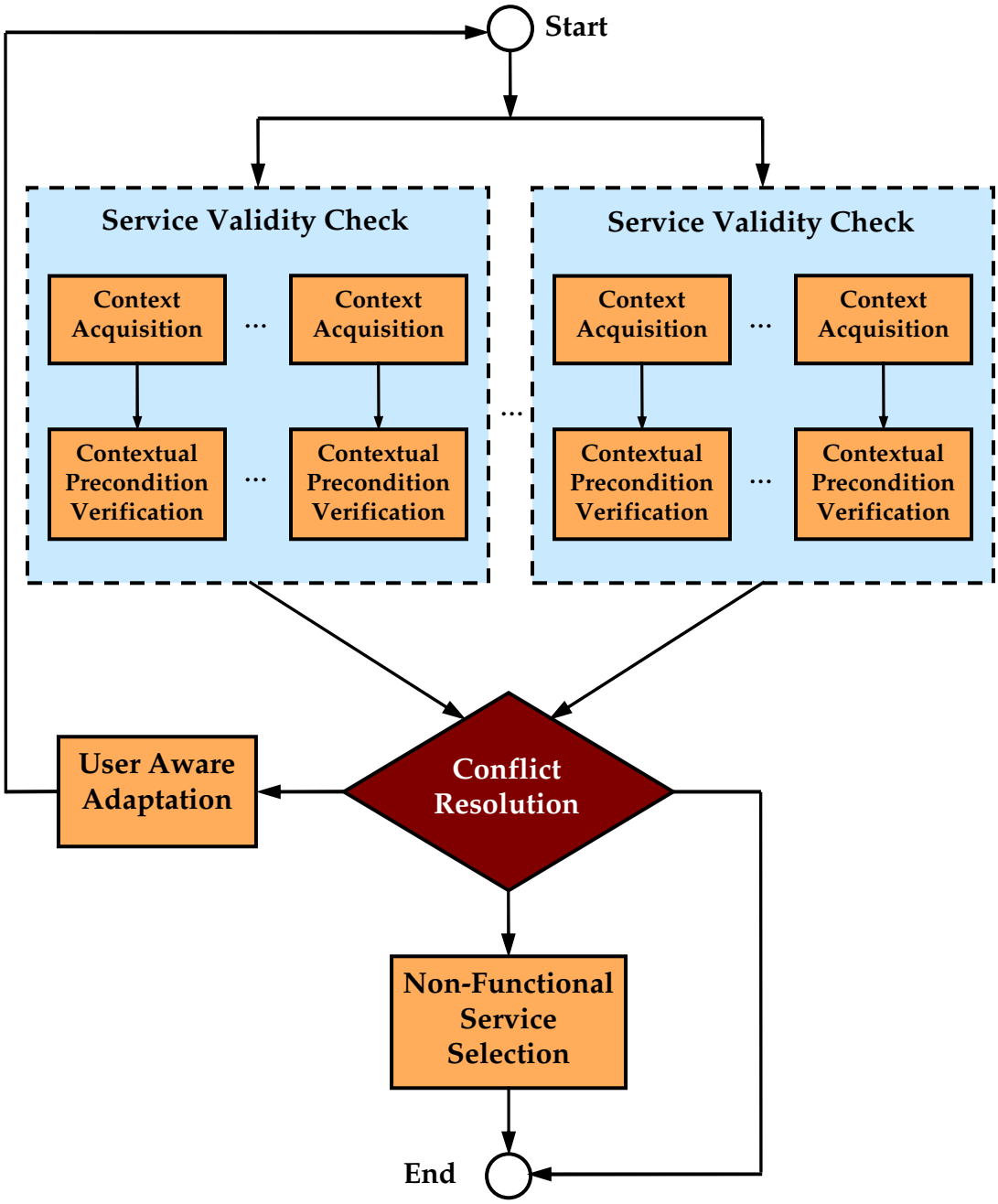

FIGURE 93 Contextual Control Functionality

In the following sections devoted to the control nets modeling using Petri nets formalism we will show some specific Petri net constructs to be used for effective construction of the described contextual control functionality.

4.4.1.3 Inter-layer Linkage

As soon as both the service net and the control nets are constructed, the entire contextual service net can be built. In order to do this, logically linked but yet previously independent service and control layers of the service model must be appropriately connected if they are to be able to perform joint operation.

The linkage between the layers is achieved by special arc constructs and using the control principle of the Meta Petri nets formalism. In brief, (for more details see Chapter 2) in Meta Petri nets the control principle is the following. Some places within the control layer are control places. They have outgoing inter-layer arcs, through which they can control associated constructs (places, transitions or tokens) from the lower layer. Whenever a token is present in one of these control places, the associated Petri net element is enabled on the lower layer of the net. Otherwise, it is disabled, i.e., removed from the lower layer net with all its incident arcs (in case of a token only the token is removed). So, inter-layer control arcs perform the mapping from presence/absence of a token in the metaplace to presence/absence of the associate place or transition in the lower-layer net.

We apply a very similar principle for linking the layers of the service model and for performing service net reconfiguration on the fly. In order to achieve that, we introduce a set of additional inter-layer arcs connecting metaplaces from the control layer to the transitions on the service layer. Note, that we use the type of transitional Meta Petri nets, because they better suit for controlling mutual exclusion constructs due to the fact that within such a construct each alternative service has individual transitions but no individual places. It is easier and more logical to control these transitions to engage and disengage component services on the service layer.

The major distinction of our control approach is that we propose "negative" control metaplaces instead of common "positive" control metaplaces. Negative control means that the controlled constructs are enabled when the corresponding metaplace is empty, and disabled when it stores a token. Such modification is justified by reliability reasons. If a contextual control suddenly fails during execution, enabled transitions preceding alternative services will allow controlled exclusion construct to act as a non-deterministic choice, which still leads to service completion. Otherwise, if these transitions were disabled by default, any control layer failure would possibly result in inoperability of the entire composite service.

Definition 4.12. A *contextual service net* is a tuple $CSN = (SN, S_{CN}, IL)$, where

- SN is the service net;

- S_{CN} is the set of control nets of the control layer;
- IL is the set of directed inter-layer arcs such that $IL \subseteq (P_{CN} \times T_{SN})$.

The general structure of the context-aware composite service with established inter-layer linkage (thick dashed red arrows) is shown in Figure 94.

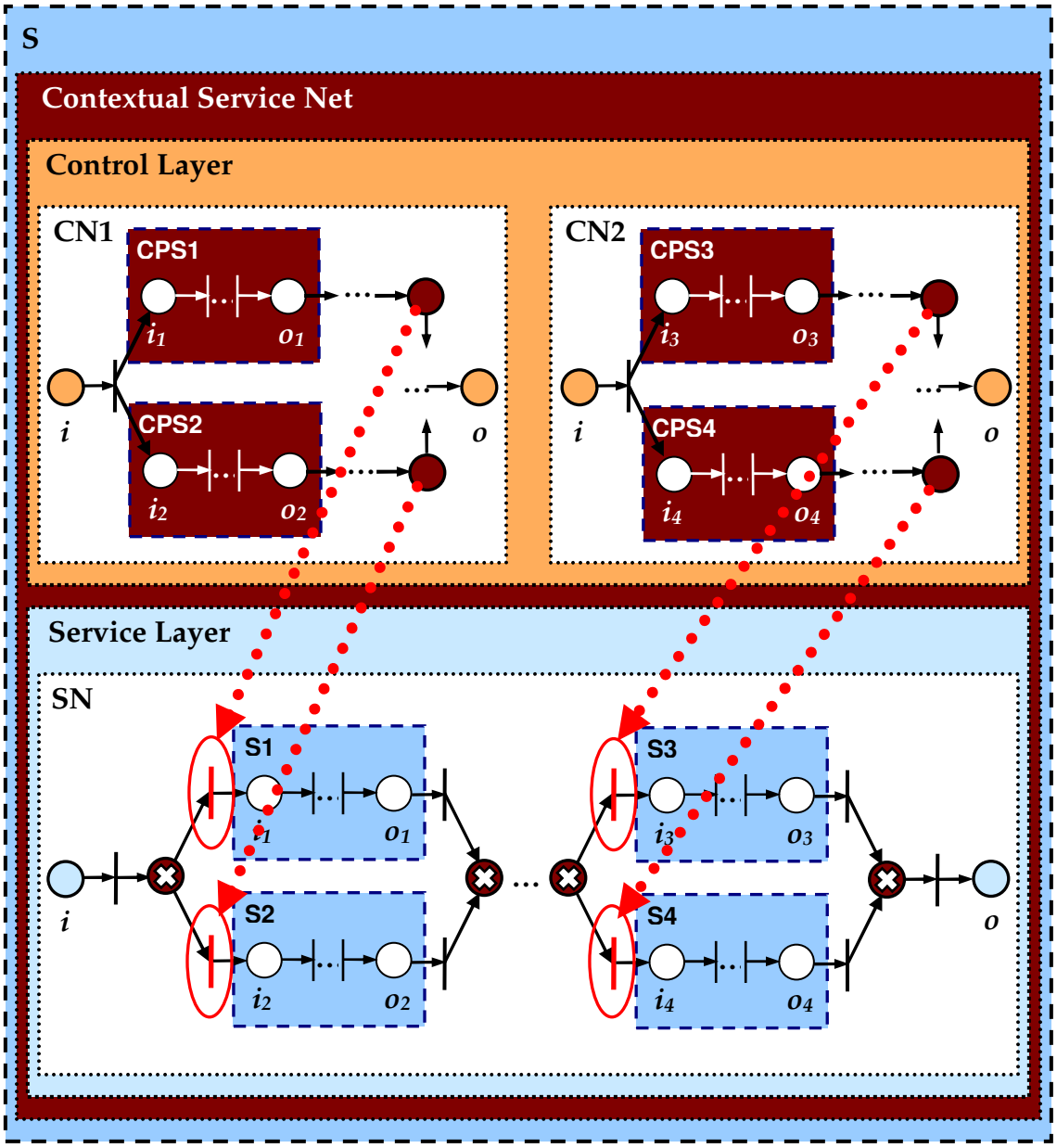

FIGURE 94 Linkage of layers within contextual service net

However, the variant of inter-layer linkage shown in Figure 94 corresponds to the basic Meta Petri net control principle: controlled transitions should be disabled until the meta-places controlling them are filled with tokens. If we revert this principle, then the transitions are enabled by default and the exclusion operator,

which embraces them, can always perform a non-deterministic choice without waiting for control actions from the control layer. But even without a reversion of the control principle, such linkage as shown in Figure 94 is far from optimal. We draw this conclusion because with such a linkage it is hard to synchronize multiple control influences coming from the control layer to the same exclusion operator. In some cases two meta-places, which control two mutually exclusive transitions on the service layer, might receive their tokens at different times, and the corresponding exclusion split would simply choose the transition which were enabled first. The other transition is also enabled, but a bit later, when it is already impossible to reverse the choice made, since the execution of the service net has already proceeded further via the first chosen transition. Therefore, either the meta-places corresponding to exclusive transitions should be updated simultaneously, or the control principle must be given a serious revision.

We select the latter option and reconsider the control principle as follows. Each control net is associated with a single controlled exclusion split within the service net. Each control net has a starting place and an ending place. Our idea is to exchange a token between the exclusion split and the control net in order to synchronize the control over exclusive transitions. So, as soon as a token comes to the exclusion split place within the service net it is transferred to the starting place of the corresponding control net to immediately initiate the contextual control procedure. If the transitions are disabled by default, there should not appear any conflict. As the control procedure proceeds, the control meta-places can be filled with tokens, and their corresponding transitions can be enabled. However, this will not result in an immediate choice between the transitions because the split place lacks a token to proceed with selection. The token is returned to the split place from the ending place of the control net as soon as it arrives there. Thus, we ensure that all exclusive services can be validated prior to the actual choice among them and regardless the time needed to complete their validation. The described control principle is illustrated in Figure 95. The blue dashed arrows connecting the two layers of the service model perform the token transfer function between the service net and the control net. So, the execution of the control net is nested into the execution of the service net, since they use the same token for going through their structure. This type of control principle has the drawback of serial execution, i.e., it is not possible to perform the reconfiguration of the service net in parallel to its execution. More specifically, the exclusive services cannot be validated and approved/discarded from the service net structure until the token comes to the corresponding exclusion split place. Though it can be perceived as the performance decreasing factor, we have to note that building a proactive control principle, which would allow performing contextual control concurrently with the service net execution and in advance to the start of the execution of the corresponding controlled exclusion split, would be somewhat difficult and often even inefficient. The difficulty with proactive control is explained by the fact that it is frequently

very hard or even impossible to anticipate by which execution branches the execution of the service net will go. Inefficiency refers to the situation where such prediction produces fallacious results, which would lead to useless control procedures' execution and extra costs to a service consumer.

4.4.2 Modeling of Contextual Control Layer

In this section we show how to precisely model the contextual control layer functionality of the service model in terms of Petri nets.

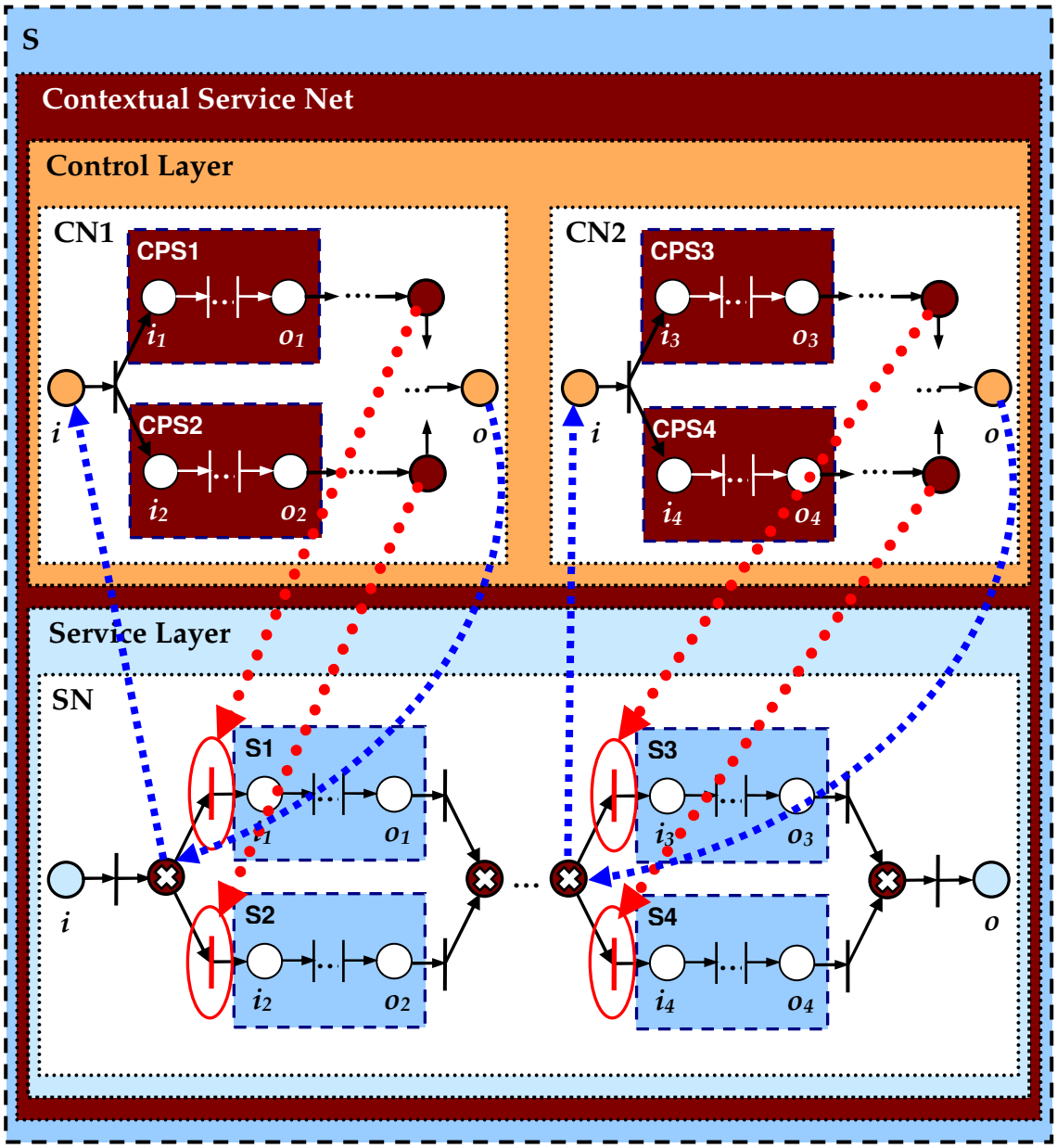

FIGURE 95 Linkage of layers based on revised control principle

It is important to notice that by modeling of the contextual control layer we mean modeling of a single control net, because as we already mentioned individual control nets are not related or interconnected within the control layer. So, the control layer is basically a collection of individual and independent control nets. Therefore, we will build a Petri net model of a single control net, which suffices for comprehensive representation of the whole control layer.

For the representation of control nets we use ordinary Petri nets, which allow concurrencies, multiple arcs and token duplication, but do not allow for conflicts. Further, we will show in this section why we need multiple arcs and token duplication. Conflicts are not basically allowed because contextual control itself should be seen as precision, a sort of robust functionality which may not produce any uncertainties. In addition, we extend ordinary Petri nets apparatus with inhibitor arcs, which are an invaluable tool for modeling logical inference on binary and cardinal data.

Starting from the description of the general structure of a control net, we further split our discussion into modeling of two main sub-procedures of contextual control: service validation and conflict resolution. We also separately discuss three specific semantic constructs we used to extend the basic Petri net formalism: semantic transitions, user actions and non-functional selectors.

4.4.2.1 General Structure of Control Net

The general Petri net structure of a control net closely resembles the functional structure we previously presented in Figure 93. However, its Petri net implementation has some specifics worth mentioning here.

First of all, a control net is always a Petri net in a standard form, i.e., it has one starting place and one ending place. Nevertheless, it allows looping, as can be seen in Figure 96, which describes the general Petri net structure of a control net.

Running a few steps forward, we say that all sub-procedures presented in a control net are also Petri nets in a standard form, though they often have some additional data inputs and outputs.

As can be naturally concluded from our last statement, we also model some data manipulations within the limits of control net modeling. Indeed, we model the necessary data inputs, outputs and inference actions based on relevant data. However, it does not go as far as modeling of full value data flow between control net operations. We model data operations only to the extent that is absolutely necessary for effective branching of the control flow and for producing grounded decisions and desired effects. We will further discuss this issue in sections devoted to component sub-procedures.

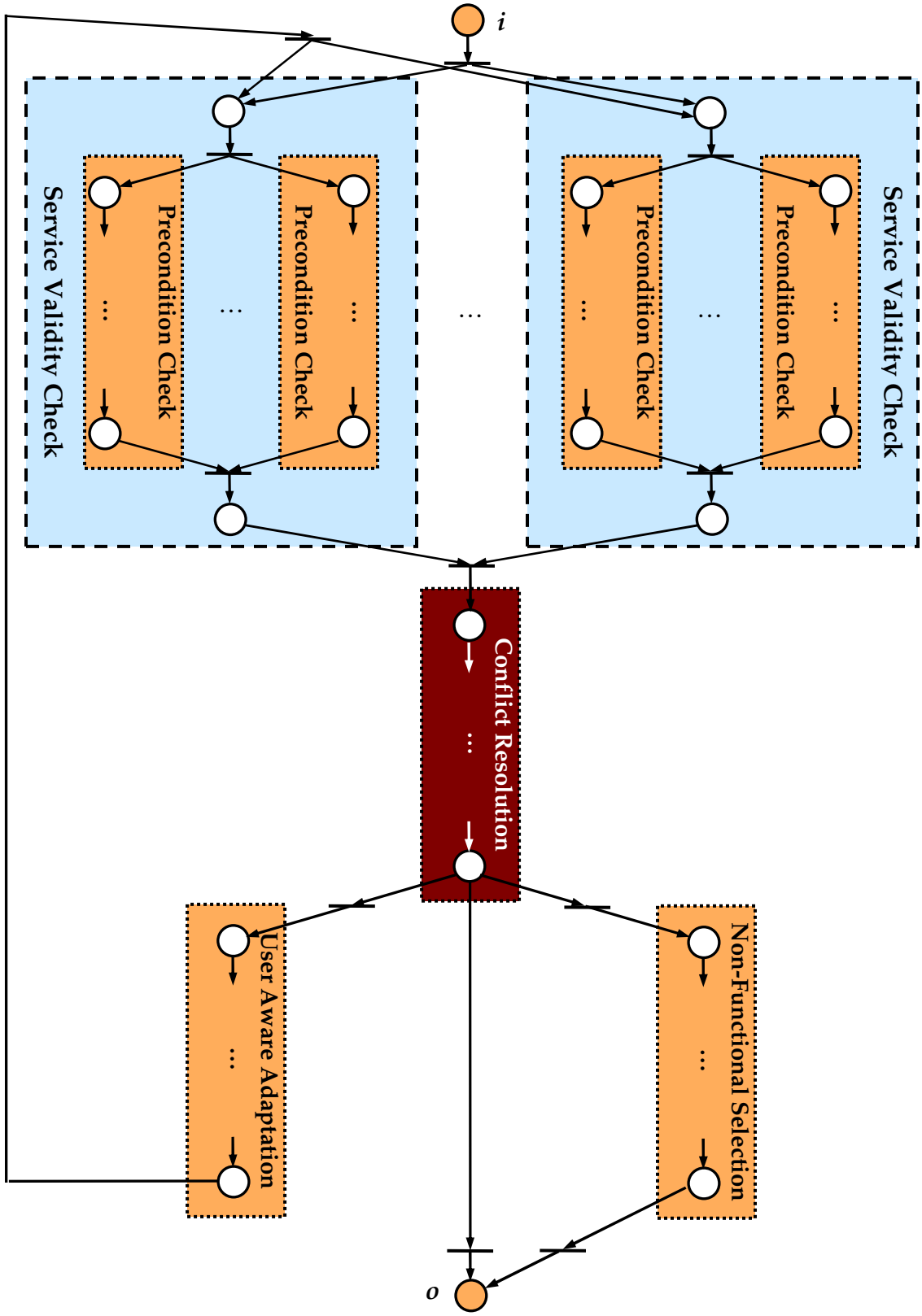

FIGURE 96 General Petri net structure of a control net

Validation procedures for exclusive component services are performed concurrently, and so are precondition check procedures within each service validation procedure. This concurrency is achieved by inserting nested parallel spits into the control net.

The conflict resolution sub-procedure acts on the base of service validations and can produce three different decisions for service selection.

Though these three decisions are shown in Figure 96 as conflicting output transitions of the conflict resolution procedure, this does not mean that conflicts are allowed. This particular conflict is resolved within the internal Petri net structure of the conflict resolution procedure.

We will further separately describe the Petri net models of the service validation and conflict resolution procedures. The procedures of user aware adaptation and non-functional service selection are modeled as atomic transition-like Petri net constructs in our framework. We will also discuss these constructs in separate subsections.

4.4.2.2 Service Validation Procedure Modeling

The goal of the service validation procedure is to answer the question "Is the corresponding component service currently suitable for inclusion into the service net structure?" In other words, the role of this procedure is to validate a service by verifying its real-time contextual preconditions.

In order to succeed, a service validation procedure has to acquire necessary real-time context from all contextual real-time preconditions of the corresponding service and actually check whether these preconditions hold with respect to the acquired context.

A service can be validated only if all its preconditions are checked. The logic is simple: the service is declared eligible for inclusion into the service net structure if and only if all its preconditions are satisfied in current context. This rule can be described by the formula:

$$V(S) = \bigwedge_{i=1}^{N} Pc(S)_i \tag{4.13}$$

where $V(S)$ – service validity function and $Pc(S)_i$ – service precondition.

A Petri net realization of such logical procedure for the case of the service having two preconditions is presented in Figure 97.

In that figure a Petri net in a standard form is shown. However, its standard form is not strict because of the Petri net's two more output places p_S^1 and p_S^0. These two places model the output of service validity function (4.13). If one of them contains a token, it means that the corresponding validity function produced a true of a false value respectively, i.e., a token in the place p_S^1 signifies that the service has qualified for inclusion into the service net, and a token in the place p_S^0

signifies that the service has failed to do that. Needless to say, these two places are exclusive due to the respective Petri net organization. They are supposed to be used as control meta-places at the stage of layers' linkage.

The reasoner constructs a Petri net of the service validation procedure in the following way. It takes the semantic service description of the alternative component service and identifies its contextual preconditions to be checked.

Then the reasoner associates each real-time precondition with a specific semantic matchmaking transition, which is discussed below. After that the reasoner discovers appropriate context provider services for each precondition or composes such services if necessary. As soon as the context provider services are found and delegated, the reasoner connects their outputs (both control and data) to the inputs of the respective semantic transitions. It subjects the inputs of the context provider services to a parallelism operator, so that the preconditions can be checked concurrently.

The role of semantic matchmaking transitions (we will call them "semantic transitions" from here on) is to model actual precondition check facilities. We will investigate semantic transitions in more depth in the next section. Here we only say that semantic transition takes context as an input and selects between two places to place a token into. One place corresponds to the case where the precondition is satisfied, the other to the case where the precondition is failed.

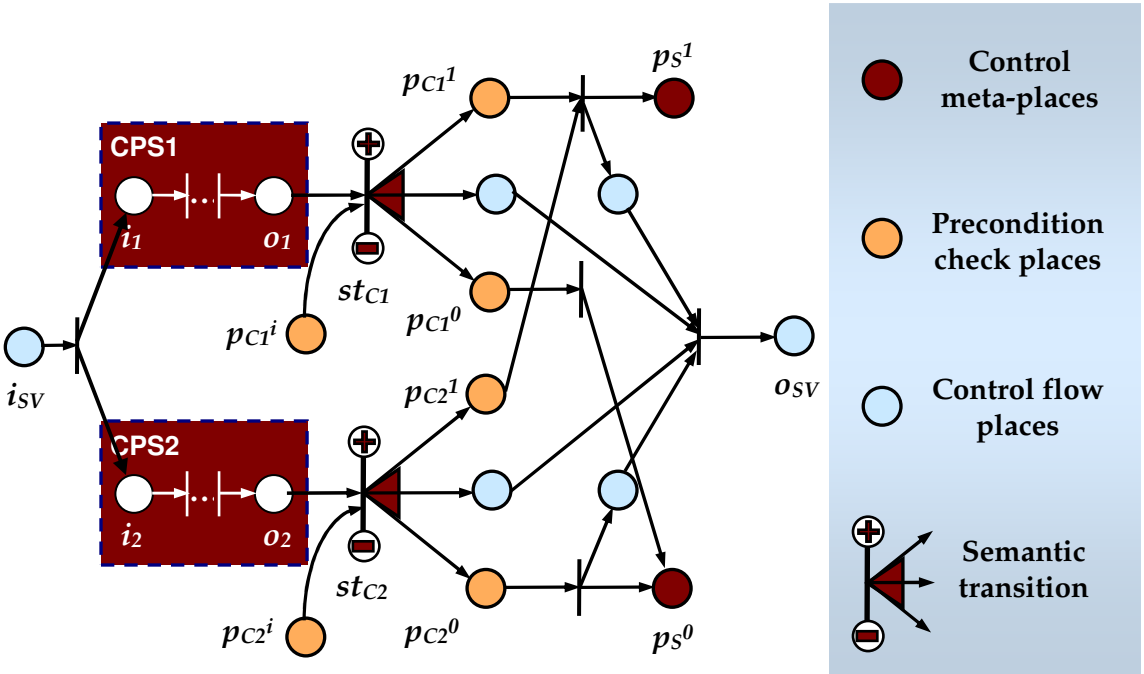

FIGURE 97 Service validation procedure for a service with two preconditions

After all preconditions have been checked, final part of the Petri net structure performs the service validity function calculation and places a token into one of

output places p_S^1 and p_S^0. When the token arrives at one of these places, the procedure can be finalized.

An interesting observation is that preconditions are generally not always conjunctive. They can be disjunctive as well. In such a case the layout of a Petri net proposed in Figure 97 does not fit for service validity checking.

We present two different Petri net layouts of service validation procedure for the cases of OR-disjunctive and XOR-disjunctive preconditions in Figures 98 and 99 respectively. We mentioned only three basic types of relationships among preconditions, namely, conjunctive, disjunctive and XOR-disjunctive. We selected these particular relationships mainly for the purpose of example. We however believe these types of relationships are sufficient for modeling of precondition logic in the majority of cases. Nevertheless, if more additional or more complex relationships have to be modeled nothing restricts that, since our Petri net modeling formalism is expressive enough to model relationships of first-order logic.

The relative drawback of the service validation procedure in such Petri net representation is that it may poorly scale with the growth of a number of preconditions, and especially for the cases where conjunctive and disjunctive preconditions are mixed.

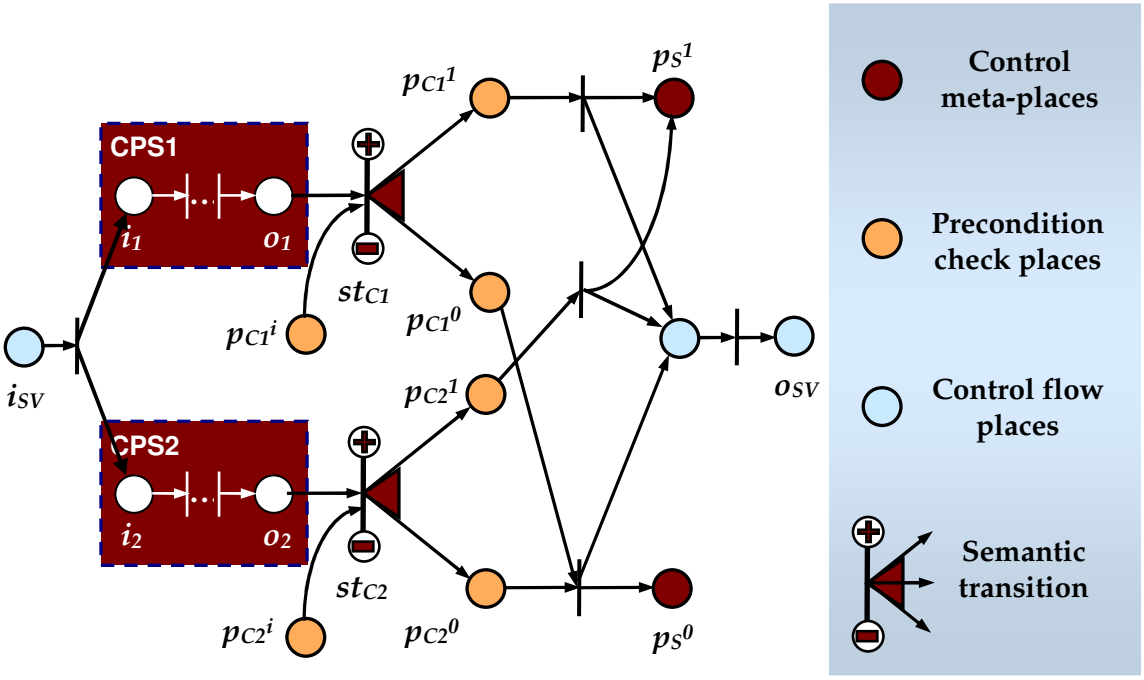

FIGURE 98 Service validation procedure with two OR-disjunctive preconditions

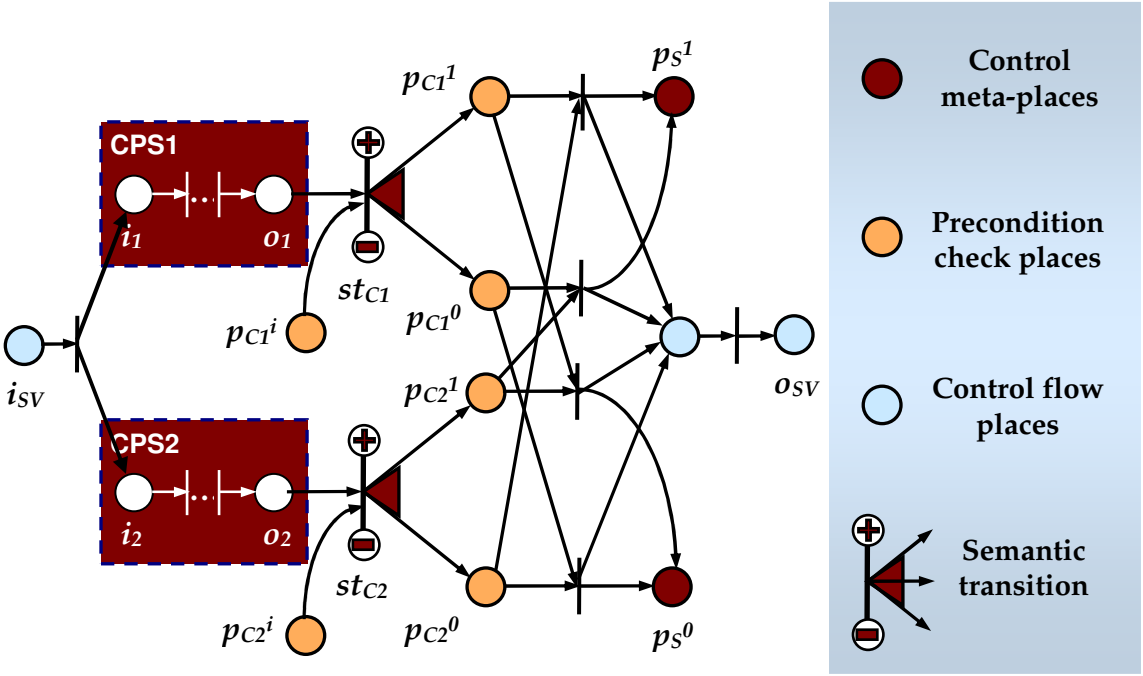

FIGURE 99 Service validation procedure with two XOR-disjunctive preconditions

4.4.2.3 Semantic Matchmaking Transitions

Definition 4.13. Semantic matchmaking transition is a special Petri net construct used for the modeling of an arbitrarily complex logical function.

In our case such a logical function is a comparison function performed by the composition reasoner. The function takes as its arguments a precondition to be verified and (a set of) context variables subjected to this precondition, and produces a logical value {0,1} as the result. This logical value indicates whether the contextual precondition is true in the current context or not. The value of this logical function is derived by the following formula:

$$Pc(pc, C) = \begin{cases} 1, & pc \subseteq C, pc \in S \\ 0, & otherwise \end{cases}, \tag{4.14}$$

where $Pc(pc,C)$ is the precondition checking function, pc is the precondition and C is the set of acquired contexts.

The precondition checking function is basically implemented as one of the reasoner's inherent functions.

Semantic transition is the analogue of the precondition checking function (PCF) in the framework of Petri net representation. The graphical representation of semantic transitions may vary and is one of those presented in Figure 100.

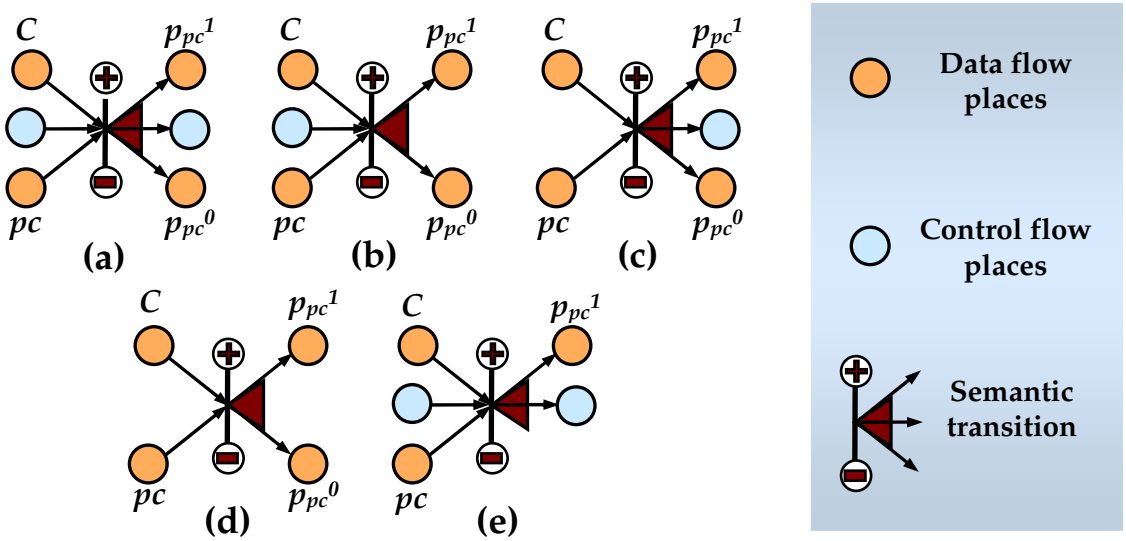

FIGURE 100 Variants of semantic transition representation

As can be seen from Figure 100 (a), semantic transition basically has three input places and three output places. One input and one output place are control flow places. They transport the execution control token through the net. They can be absent from the layout of semantic transition if some of data places also take the role of control places, such as shown in different variants (b)-(d) in Figure 100.

Data input places model the flow of context (place *C*) and precondition (place *pc*). Data output places model two possible values of PCF: p_{pc}^1 – for "1", and p_{pc}^0 – for "0". However, the latter output place can be often reduced as shown in the variant (e) in Figure 100. Such reduction can be made if zero-testing of the result of the precondition check is never applied within the control net structure. However, it is easily seen in Figures 97–99 that we use these negative outputs for determining services' validity, both positive and negative. We will further discuss the usefulness of modeling of negative service validity in the next section.

So, semantic transition basically acts as a disjunctive transition, though ordinary Petri nets allow only for conjunctive transitions. Nevertheless, taking into account that we model a service as a combination of other services and pieces of functionality such deviation from the conventional layout of Petri nets does not have a significant impact on our modeling framework. We utilize semantic transition concept mostly to denote the corresponding logical functionality of the semantic reasoner.

4.4.2.4 Conflict Resolution Procedure Modeling

Conflict resolution procedure starts its operation as soon as all service validation procedures have been completed. The objective of the conflict resolution procedure

is to detect conflicts among validated component services and to decide how to resolve a conflict if one appears.

We should clearly distinguish between two different concepts of "conflict" we use in our discussion. One conflict concept refers to a situation where a Petri net contains a place, which has two or more outgoing arcs, and at least one of these arcs shares its destination transition with no other arcs. So, a conflict is a state of affairs brought up by the presence of more than one mutually exclusive paths in the state space and the choice among these paths is in no way determined. The essence of conflict is then in the fact that it cannot be determined through which arc a token will go from this place.

The conflict concept we use in the context of conflict resolution procedure has nothing in common with a non-deterministic choice type of conflict. Here by conflict we mean the situation where it cannot be sharply determined which one of the validated services to select.

This is quite an important task since we have to carefully choose only one service from among all possible alternatives. If two services have been validated, this is unacceptable and the choice between them still has to be made. So, the conflict resolution procedure aims at selecting exactly one alternative service within the given set of validated alternatives.

The Petri net model of the conflict resolution procedure is presented in Figure 101.

More specifically, three different scenarios for conflict resolution are possible.

1. *Single service scenario.* This is the easiest kind of situation. Only one alternative service has qualified as the result of service validation to be included into the service net structure. No additional selection efforts are to be done.
2. *Multiple services scenario*. Can also be called double service scenario since the number of qualified alternative services makes no difference from the viewpoint of conflict resolution, if this number is bigger than one. In this case an additional non-functional service selection procedure must be utilized to discriminate between the services.
3. *No service scenario*. The most difficult type of scenario. No alternative services have been qualified during service validation procedures. So, nothing can be put into the service net. Such situation can only be managed by the user, who can undertake certain actions in order to change context. After that services can be re-validated.

As it can be seen in Figure 101, the conflict resolution procedure has three data outputs, namely, P_{CR}^1, P_{CR}^2, and P_{CR}^0. P_{CR}^1 corresponds to the single service scenario, P_{CR}^2 to multiple services scenario, and P_{CR}^0 to no service scenario respectively. The task of the conflict resolution procedure is, thus, to determine which scenario currently takes place and mark its respective output.

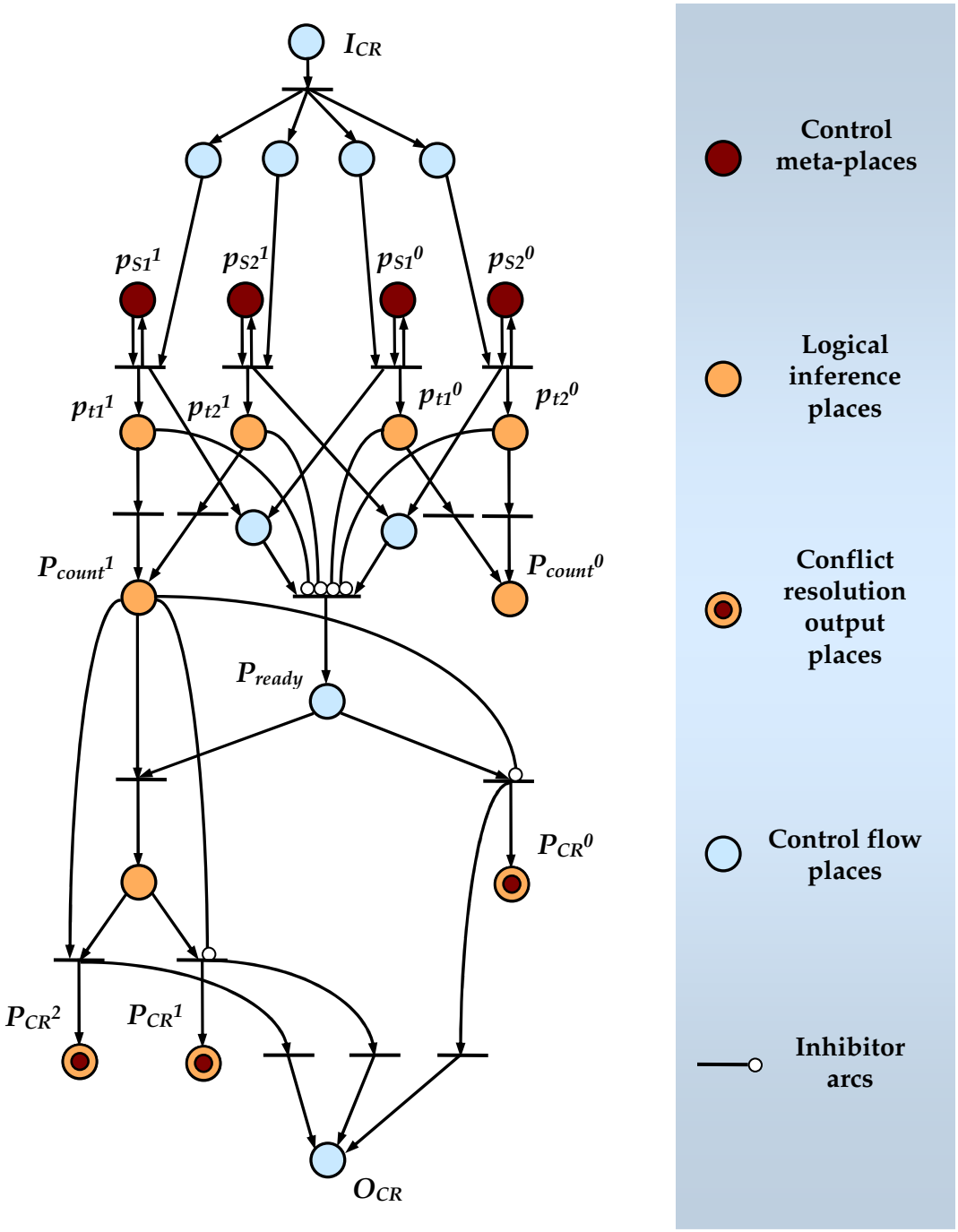

FIGURE 101 Conflict resolution procedure.

The conflict resolution procedure operates as follows. It takes as its inputs control meta-places previously filled by service validation procedures. Then it copies tokens from all marked control meta-places $P_{Si}^{(0,1)}$ to their temporary analogues $P_{ti}^{(0,1)}$ without changing the marking of the meta-places. After that the procedure computes the number of qualified and non-qualified services by accumulating

corresponding numbers of tokens in counter places P_{count}^1 and P_{count}^0 respectively. To make sure that all services are counted, the procedure uses a special check place P_{ready}, which can be marked only after the start of counting process and is actually marked only when all $P_{ti}^{(0,1)}$ places are again empty. Note that the procedure uses inhibitor arcs for zero-condition testing on this set of places.

As soon as the counting process is complete, the procedure starts actual logical inference of the current conflict scenario. If the place P_{count}^1 is empty and P_{ready} is marked, we encounter the no service scenario, and the place P_{CR}^0 is marked. Otherwise one token at P_{count}^1 is transferred to an auxiliary place, and the marking of P_{count}^1 is again tested. If it is empty, we have the single service scenario, and the place P_{CR}^1 is marked. If it is not empty, the multiple services scenario persists, and the place P_{CR}^2 is marked. As one of the data output of the conflict resolution procedure becomes marked, the procedure terminates.

Then if it is P_{CR}^1 which has been marked, the whole control net execution also completes, and the final marking of control meta-places P_{Si}^1 will be used for reconfiguring the service net structure on the lower layer. If P_{CR}^0 has been marked, the user aware adaptation procedure is started. Finally, if P_{CR}^2 is the marked, output of the conflict resolution procedure, non-functional service selection procedure, should still be initiated prior to completing the control net execution.

In the Petri net structure of Figure 101 there is one more blind place – P_{count}^0. This dead-end place models the counter of non-qualified services and is not used in any further execution of the conflict resolution procedure. A natural question may arise: why is it needed? Although we do not use this place, we argue that testing of the number of non-qualified services can be useful for certain modifications of the conflict resolution procedure and of the whole control net. In any case, we can easily throw it away from our conflict resolution procedure for now. And this will result in another logical question: why then negative service validation meta-places P_{Si}^0 are needed? Even though in the absence of non-qualified services counter we do not use these places for their direct purpose, they are still necessary for organizing the appropriate control flow of the conflict resolution procedure within the Petri net.

4.4.2.5 User Actions

User aware adaptation procedure starts if the conflict resolution procedure detects the “no service” scenario, i.e., if no services qualified for inclusion into the service net.

In this case the semantic reasoner have to devise a corrective action, a proposition to the service consumer to perform certain actions, which will result in such context changes that would allow some of the alternative services to qualify in the new context. We do not have any intention to describe how the reasoner could derive such corrective actions. We feel this issue is sophisticated enough to

dedicate a separate thesis to. Therefore, we are only interested in giving a possibility to model such intelligent feature within our Petri net-based framework.

Since we are not going to model the actual process of corrective actions derivation and conceive this procedure as another implicit functionality implemented by the reasoner, we merely represent the user aware adaptation procedure as an atomic construct – user action transition.

Then the entire user aware adaptation procedure can be brought to a single transition with some minor auxiliary Petri net elements, as depicted in Figure 102.

In Figure 102 the actual user aware adaptation procedure is embraced by the dashed rectangle. The rest of the Petri net elements are included to show the linkage of this procedure to the conflict resolution procedure and the service validation procedures. More specifically, the control input I_{UA} of the user aware adaptation procedure is linked to the control output O_{CR} and one data output P_{CR}^0 (corresponding to the "no service" scenario) of the conflict resolution procedure. The control output O_{UA} is linked to the control inputs I_{SVi} of all service validation procedures, thus forming a loop in the control net.

The user aware adaptation procedure itself consists of the mentioned input and output, a user action transition, and two data places. The data place P_{CA} models the corrective action proposed by the reasoner. The data place P_{UA} models the actual action the user has undergone. These two places are simultaneously the inputs and the outputs of the user action transition. However, this transition operates in a rather unusual way compared to normal Petri net transitions.

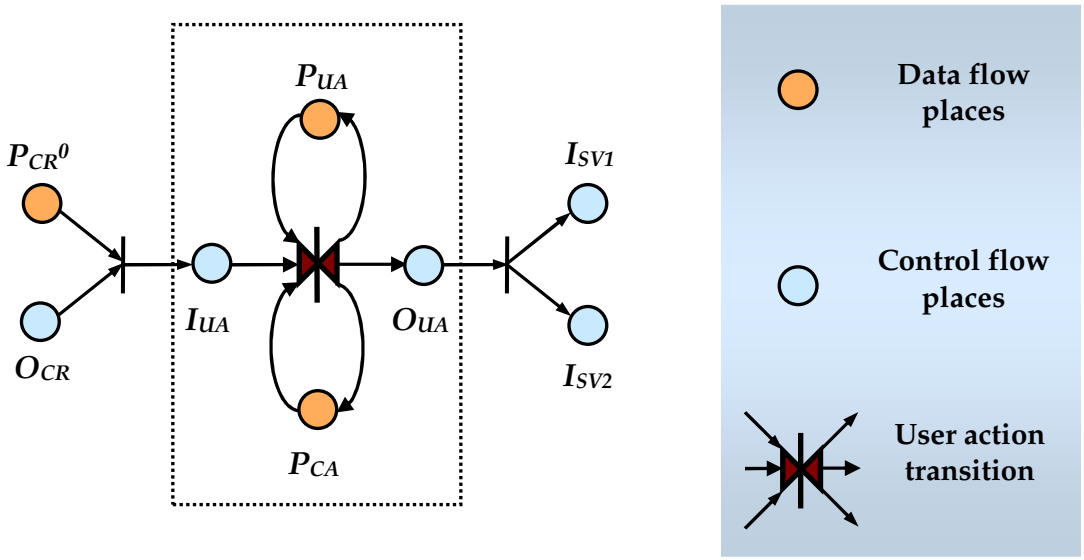

FIGURE 102 User aware adaptation procedure

The user action transition is enabled if it has a token in its control input place I_{UA}. If the transition is enabled, it consumes the token from its control input place and posts a token in the place P_{CA}, which means that the reasoner can now propose a

corrective action to the user. Then the transition consumes the token from P_{CA} and posts a token into the place P_{UA}. The reasoner waits for the appropriate action to be performed by the user and as soon as it senses that the necessary result is achieved it activates the transition again. The transition consumes the token from the place P_{UA} and posts it to the control place O_{UA}. The user aware adaptation procedure is complete.

A logical enhancement to the user action transition's operation principle can also be made. It cannot be generally expected that the user always performs corrective actions proposed by the reasoner. So, it might take an eternity to wait for his/her appropriate action. Thus, it would be natural to make the user action transition a timed transition, i.e., a transition that waits for a certain amount of time before it operates. In such a case after a certain amount of time the transition removes the token from the place P_{UA} and posts it to the place P_{CA} for the reasoner to revise its corrective action. Then the process starts over.

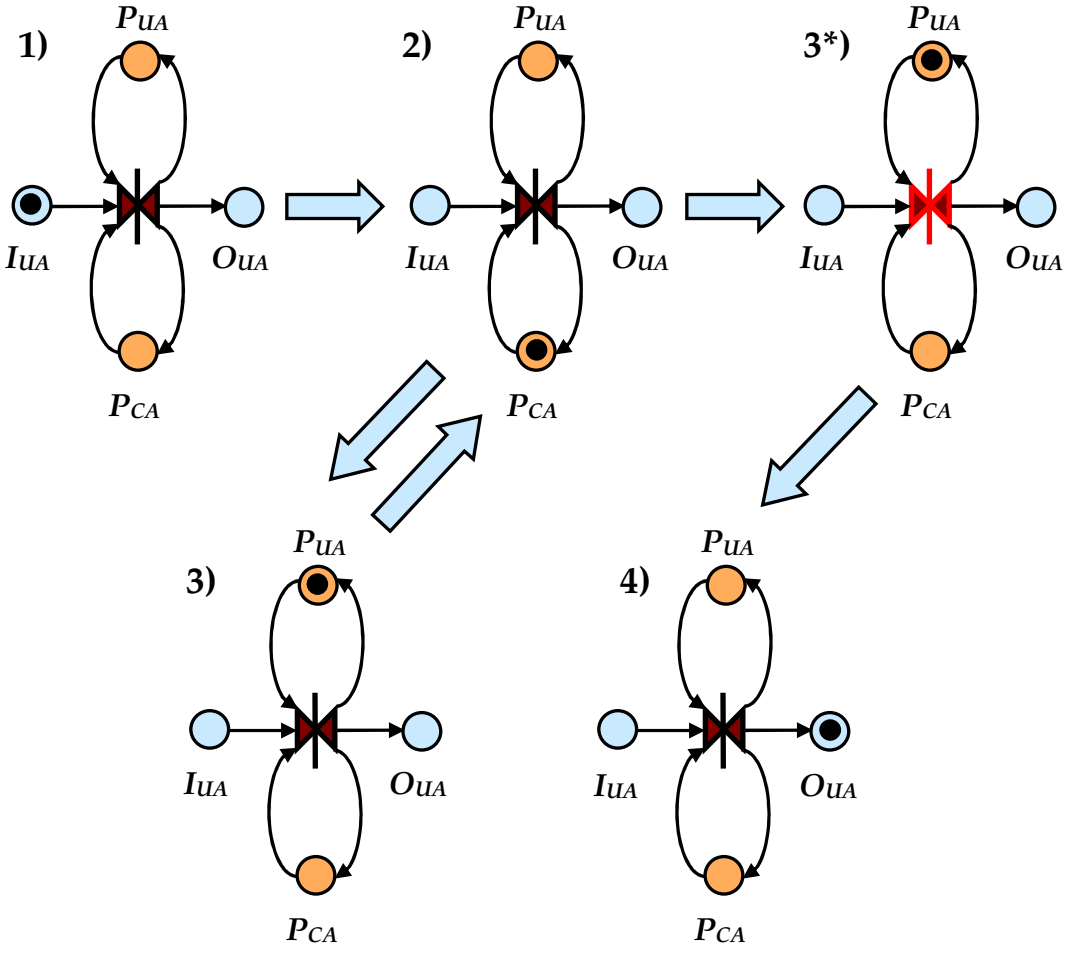

FIGURE 103 Token game of the user aware adaptation procedure's Petri net

In order to clarify the principle of the user action transition operation, we utilize the popular "token game" abstraction for the demonstration of Petri nets execution. A token game of the user aware adaptation is shown in Figure 103.

In that figure we can see five different markings of the user aware adaptation procedure's Petri net. They are numbered in the order of reachability. Markings 3 and 3* are graphically the same, but they differ in the way they activate the transition. Marking 3 corresponds to the case where user action timeout expires and the token is re-posted to the place P_{CA}, thus returning to the marking 2. The marking 3* corresponds to a case where the user has undertaken the proposed corrective action and thus activated the transition manually before the timeout expired, which consequently leads to the final marking 4. So, during the token game the user action transition is mostly an automatic one except for the case of marking 3, when the user can manually fire this transition. From our point of view, this type of token game illustrates well the operation principle of the user action transition.

4.4.2.6 Non-functional Selectors

Non-functional selection procedure can be started as soon as the conflict resolution procedure completes and if it has detected that the "multiple services" scenario is currently active.

The role of the non-functional service selection is to provide an additional selection tool for the case where functional validation and comparison of alternative services does not help determine the best choice.

So, if two or more services out of the whole set of alternatives have qualified during a service validation procedure, i.e., they totally comply with the current context, they can be discriminated only on the basis of their non-functional properties. Non-functional properties, as we already described previously, are basically various QoS characteristics such as usage cost, time to complete, hardware requirements, resource reservations, etc. Our idea is again not to thoroughly model the discrimination procedure, which operates on top of these characteristics, but to simply propose an adequate operational semantics, which would allow seamless inclusion of the corresponding system procedure into our Petri net models.

Although similarly to user aware adaptation we see the non-functional service selection as an inherent reasoner's functionality and hence it is out of our scope, we need to say some words about this type of procedure. Non-functional service selection can be organized in variety of ways and using different principles of comparison and sets of criteria. We do not, however, incline towards any of these variations and merely leave the issue to the reasoner's designers. Nevertheless, we place two major requirements to such non-functional selection procedure. First requirement dictates that the procedure has to operate over the

same set of variables, treating them both as inputs and outputs. This is absolutely necessary in order to build a sort of multiplexer functionality, i.e., a functionality that selects one input out of many and transfers it to the output. This requirement should be established because we want to operate with the set of control meta-places serving as indicators of services' validity. The final marking of this set will be used to determine the final service choice. Before the non-functional service selection procedure this set of places $p_{Si}^1 \in P^1, i = \overline{1,n}$ has multiple marked places. After this procedure's completion it should have exactly one marked place. So, the transformation must be the following:

$$\sum_{i=1}^{n} \mu\left(p_{Si}^1\right) > 1 \xrightarrow{NFS} \sum_{i=1}^{n} \mu\left(p_{Si}^1\right) = 1 \tag{4.15}$$

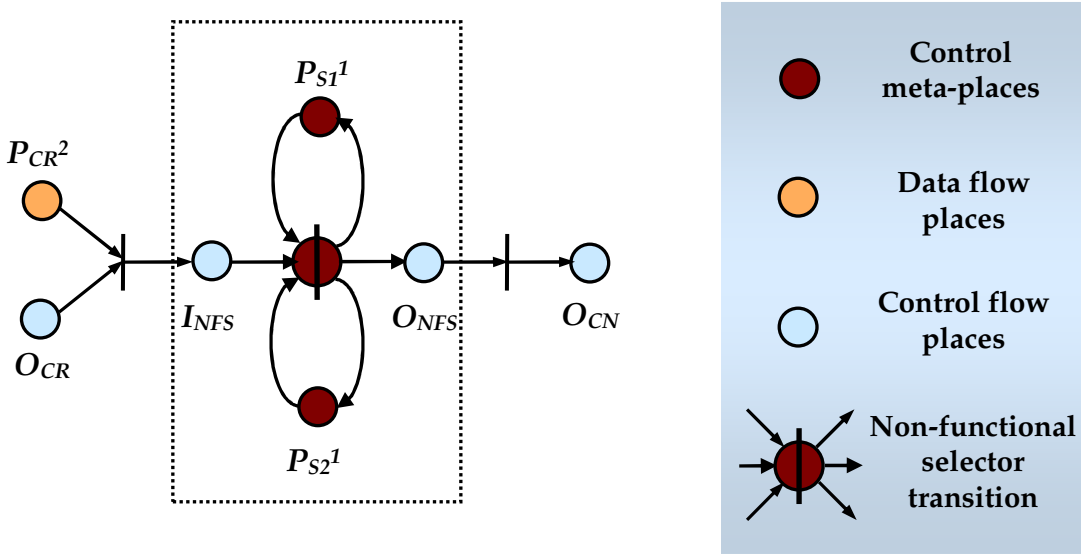

FIGURE 104 Non-functional service selection procedure.

In terms of Petri nets such transformation can be performed by a transition, which has the same set of n input and output places. The transition is enabled if at least one of its input places has a token (according to our control net organization there will be at least two such places). On firing, the transition removes tokens from all marked input places, while ignoring all empty places, and deposits a token into only one output place, which was necessarily among the marked places before the transition fired.

We call such transition a non-functional selector transition and illustrate its structure and the layout of the non-functional service selection procedure in Figure 104. In that figure we can also see that the non-functional selector transition has also one more control input and one more control output. The non-functional service selection procedure starts when the control output O_{CR} and the data output $P_{CR}{}^2$ of the conflict resolution procedure have tokens. Figure 104 shows the most

primitive case of the non-functional selector transition, which has only two data inputs. It is trivial because according to our control net organization the non-functional service selection procedure is activated if and only if two or more services have qualified during service validation. Since the depicted non-functional service selector transition has only two data inputs (which means that in total two services compete for inclusion into the service net), they both are marked, by definition. The case of three and more alternative services is much more interesting because it is possible that not all of them have qualified during service validation, and the transition will potentially have empty inputs. However, we already described in the previous paragraph how this transition would behave under such conditions. As the non-functional selector transition successfully fires, it deposits another token into the non-functional service selection procedure's output place O_{NFS}, which in turn leads to a successful completion of the whole control net.

Another requirement we place on the non-functional service selection concerns the principles of how this selection is being made. We feel that it is important to prioritize evaluated non-functional properties of the compared services with respect to given user preferences, should they be provided.

Though the service selection principle can be arbitrary and is basically beyond our scope, we can describe the general idea of how the non-functional comparison of services is to be made. Based on the given user profiles, which contain QoS preferences among other things, the reasoner selects important (from the user perspective) non-functional characteristics, prioritizes them with respect to the given preferences, analyzes them, transcribes them into some numerical equivalent if needed, and finally calculates certain linear (non-linear can be also the case) characteristic function for each compared service with appropriate prioritized weighting of component arguments. The service with the highest score is then selected for inclusion into the service net.

4.4.2.7 Control Net Execution

In this section we highlight some aspects concerning the structure and the execution of the whole control net. The entire control net structure is shown in Figure 105. This structure has a graphical representation that is too sophisticated to allow us to discuss its execution in a step by step fashion. So we simply emphasize some issues which should be paid attention to.

A control net is a Petri net model in a standard form. This is reflected by the fact that a control net always has exactly one starting place I_{CN} and exactly one ending place O_{CN}. However, for this type of Petri net structure we allow several indulgencies in formal requirements in order to increase modeling flexibility and efficiency. Namely, we allow control nets to be non-conservative: the tokens can be duplicated and reduced at certain sites of a net. The main reason for this is that we model both control and data flow on the partially the same sets of places and

transitions, so places often can act both as data and control states, and data and control tokens must be merged. Also, we implement counters in our control net structure. Counters naturally duplicate tokens.

In addition to token duplication, we utilize loops, and inhibitor arcs. Loops are needed in order to revise services' validity if all of them failed to qualify. Inhibitor arcs are powerful for data flow modeling, because they facilitate zero-condition testing. Inhibitor arcs are especially useful when treating counters for transcribing integer values into logical values.

Despite all these extensions to ordinary Petri nets formalism, we construct our control net models in such way which ensures that the net is live and the final marking is always reachable.

Speaking of the execution of the Petri net model shown in Figure 105, we have to note that this Petri net is a simple linear combination of component procedures' Petri nets described in previous sections. The order of their execution is set by the control flow structure (marked with blue places). All procedural subnets are executed as described in their dedicated sections. The control net selects between two alternative services, one of which has two conjunctive preconditions and the other two XOR-disjunctives.

The red dotted lines in Figure 105 indicate that the pairs of places they connect are actually one place, but we intentionally put two instances of each in order not to complicate this complex figure any further.

4.4.3 Metanet Construction

We already discussed the major issues of linking control and service layers into a Meta Petri net in Section 4.4.1.3. Here we briefly describe some linkage issues specific for and hardly understandable without the detailed control net representation we made above.

As we already mentioned, the control net can start as soon as its starting place receives a token from the corresponding exclusion split place on the service layer. Then this token goes through the whole control net and reaches the control net's ending place. As soon as the token arrives in this place it is transferred back to the corresponding exclusion split place on the lower layer.

However, the control net has more output places than simply the ending place. We call these places "control meta-places" and denote them as P_{Si}^{1} and P_{Si}^{0}. As we have investigated in the previous section, these places encode facts about whether the corresponding alternative service S_i has either qualified for inclusion into the service net or not. So, these places always unambiguously indicate whether the corresponding service can be used within the given service composition structure or not. That is why they should be used as control places to reconfigure the structure of the service net.

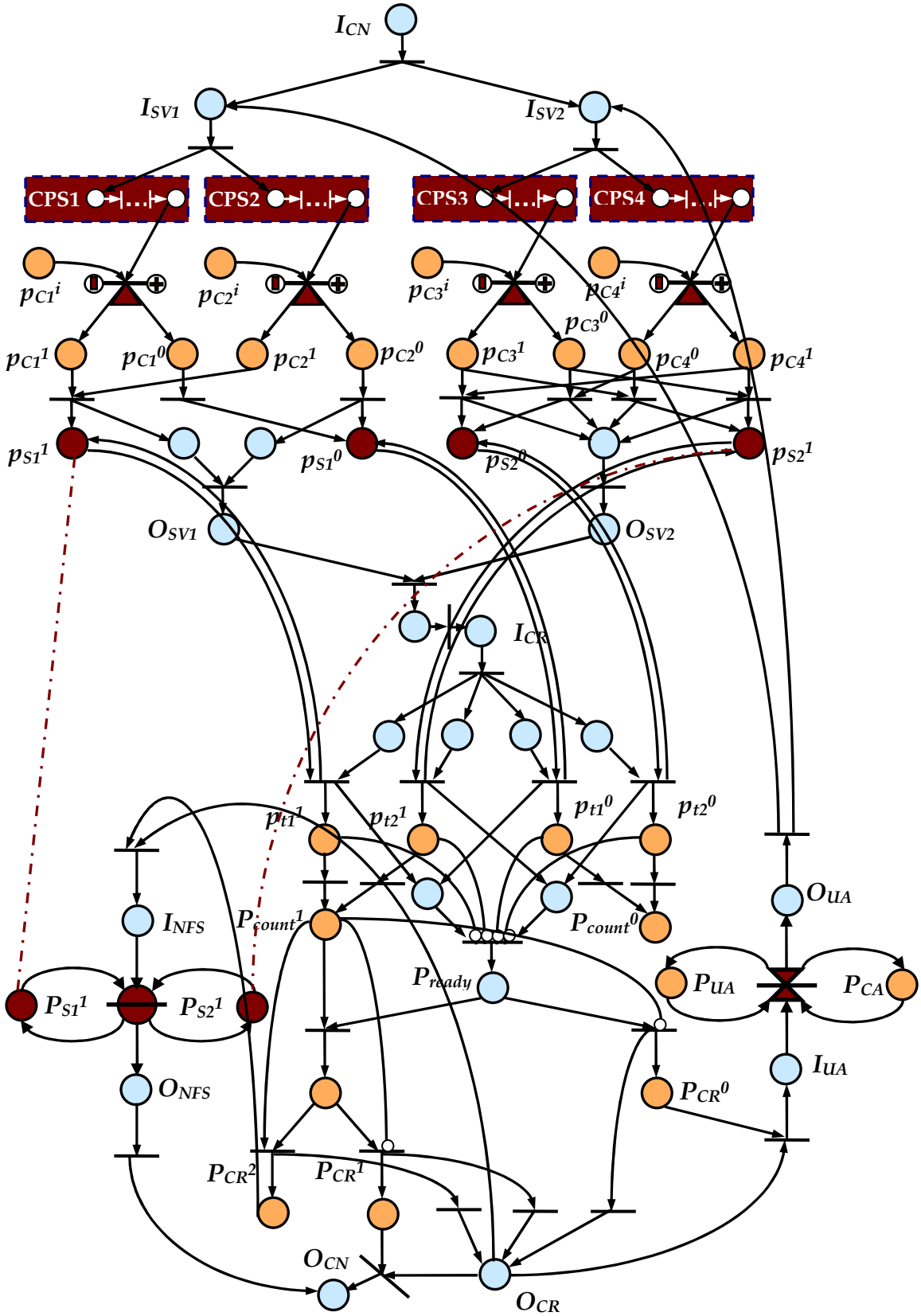

FIGURE 105 The final structure of the control net

It is obvious that places with the same lower index (S_i) and with different upper index (0 and 1) form a pair. Such pair exhibits apparent redundancy. If one of the places does have a token, then the other one definitely does not, and vice versa. So, in principle one place out of two is sufficient for modeling services' validity. Nevertheless, we presented these pairs of control places intentionally. Having both options allows building a more flexible control framework. For example, the main control principle of Meta Petri nets can be easily reverted if we have such counter-pairs of control meta-places.

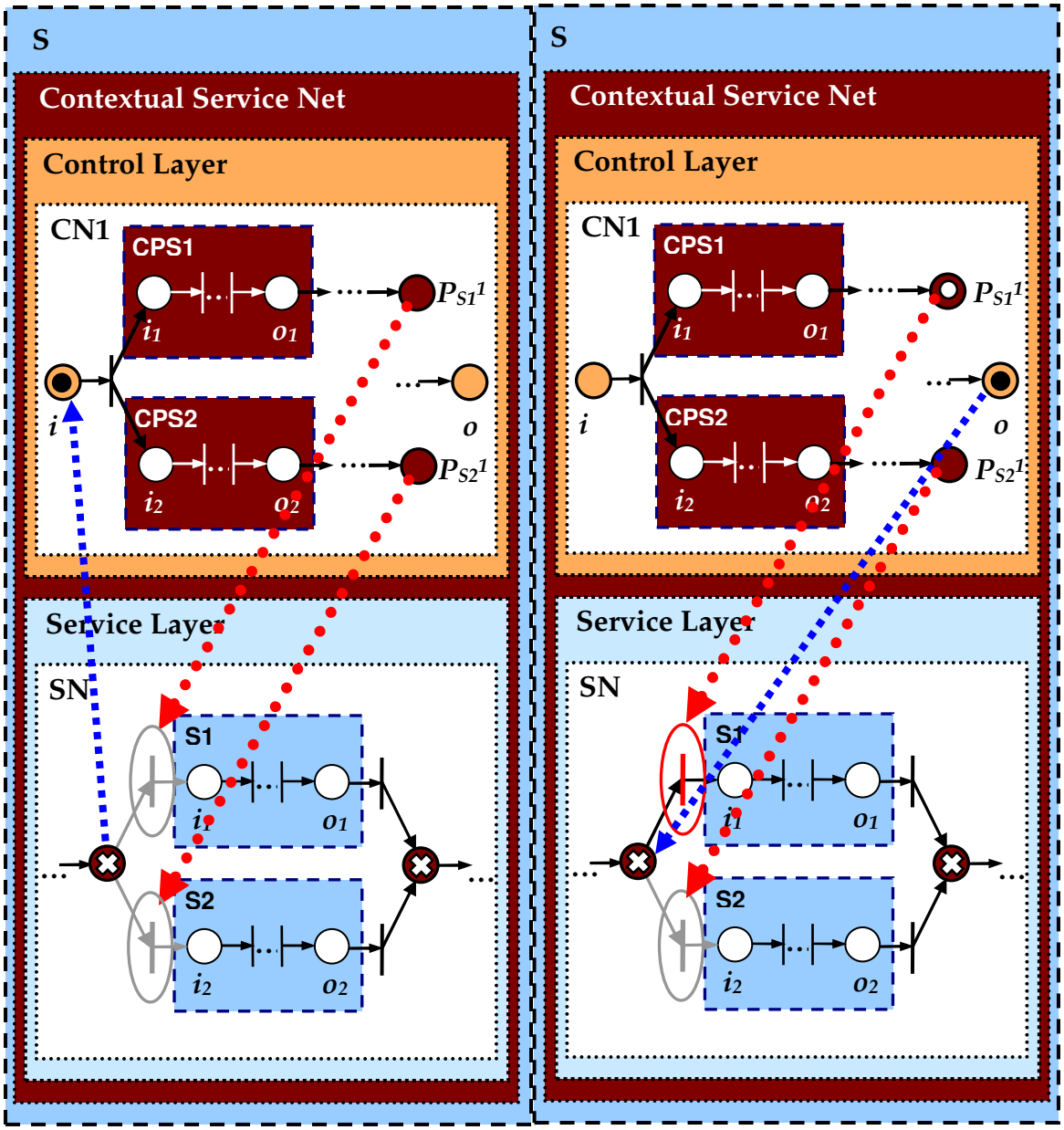

FIGURE 106 Service Meta net.

The default option in our case is nonetheless positive control meta-places P_{Si}^1. These places, when storing a token, allow enabling the controlled constructs from the lower layer net. So, using the established meta-links these places control presence/absence of the associated transitions with all their incident arcs within the service net, i.e., actually perform reconfiguration of the service net. This reconfiguration process is illustrated in Figure 106.

The left-hand side of Figure 106 shows the state of the service Meta net execution after the control net has just received a token from the service net's exclusion split. At this stage the exclusion split place's adjacent transitions are disabled because neither the place P_{S1}^1 nor the place P_{S2}^1 has a token. Their absence within the service net is indicated by gray color. On the right-hand side of Figure 106 we see a completely different stage and a completely different situation. The control net has just finished its execution, and the control flow on the service layer can be started from the point where it stopped before the control net execution. However, if it could not previously start because all output transitions of the exclusion split were absent, it can start now since one of the control meta-places, namely, P_{S1}^1 contains a token, which allows the corresponding input transition of the service S_1 to enter the service net. Thus, the choice between S_1 and S_2 is made, and the execution of the composition model may proceed further with no delays. This explanation briefly illustrates the key principle of our context-aware composition reconfiguration process.

4.5 Concluding Remarks

In Chapter 4 we achieved several important goals. First of all, we showed how to bring context as a new and valuable control dimension into current service composition practices. We realized different paradigms of context awareness simultaneously. Rather than being complete in themselves, these paradigms complement each other. Personalization is achieved via a thorough treatment of a user request, established goals and profiled user preferences. Active context awareness is the core paradigm our framework realizes, ensuring that in most scenarios it can succeed in building robust service compositions automatically and without the need for human assistance. However, in critical cases passive context awareness feature, which we call user aware adaptation, must be toggled and utilized to overcome the deadlock. Although such application of passive context awareness is rather specific compared to common practices, we feel that it might still be acceptable in terms of intrusiveness and user distraction.

We also investigated and structured the process of automated service composition in depth. Finally, we contrived and developed a formal and flexible modeling tool for context-aware service composition.

With respect to the requirements of the composition framework, which we described in Section 4.2.1, we made the following advances and achieved the following results.

- *On-demand composition*. We showed how to formalize a user-provided or automatic composition request into the appropriate composition goal.
- *Goal-driven composition*. We designed a formal framework for extraction and interpretation of composition goals, as well as the methodology for transcribing these goals into valid composition plans.
- *Context-aware composition*. We designed a service composition framework which composes services with regard to the current user and service context. The framework picks up services the contextual requirements of which can be satisfied with current context. Moreover, we not only showed how to compose services in a context-aware fashion, we also developed the formalism for modeling real context-aware composite services having their own on-board context-aware functionality.
- *Verifiable composition*. Our Petri net models of composite services are easily analyzable and verifiable regarding the correctness of their structure. Mature analysis techniques of Petri nets formalism and our thorough flawless design principles jointly contribute to the overall composition quality and robustness.
- *Reconfigurable composition*. We created a formal approach for modeling and building context-aware composite services which can dynamically reconfigure their own structure on the fly according to the changing context.
- *Automated composition*. We deeply investigated the problem of service composition automation and presented an automated composition framework, which is capable of automatic composition planning, and in particular, performs automatic goal interpretation, service discovery, service matchmaking and control flow construction procedures.

5 CONCLUSIONS

In this chapter we summarize the results obtained. We specifically give answers to the research questions formulated in Chapter 1, present an overview of the made contributions, discuss the benefits of the proposed solution and identify possible directions for continuation of the presented work.

5.1 Thesis Overview

This thesis proposes a comprehensive solution to the Web Service Composition problem, a well-recognized and overwhelmingly important challenge in modern Web and Web Services research. The problem is however considered in a non-trivial setting. This setting adds new and crucial challenges specific to Web Service composition problem along with considering traditional yet critical challenges such as automation and goal orientation. The major research and design challenge of the proposed solution can be regarded as innovative. It aspires to include context into the Web Services specification and bring it as another control dimension into the Web Service composition process. Context-dependent representation of Web Services' process models is called upon to tackle the static and largely deterministic nature of more traditional composition models. Composite process models can be made non-deterministic by subjecting them to

run-time context-aware control and become dynamic by using context-aware reconfiguration procedure.

Technically, the thesis is devoted to the design of a formal framework for Context-aware Web Service Composition. This framework is subjected to some serious conceptual and technological requirements (presented in Section 4.2.1), which ensure that Web Services are composed in an automatic and meaningful fashion to achieve user-specific goals by producing more accurate and relevant service compositions. The framework embraces all the stages of the complex Web Service composition process, from the user goal formulation to execution the resulting composite service, and identifies the usage of contextual information to enhance the functional strength and quality for each constituent procedure within the overall composition process. The framework focuses on the modeling aspect of Web Service composition, i.e., it presents the elaborate formal apparatus based on Petri nets to model operational semantics of Web Services and Web Services compositions. What is more it models Web Services in the way that allows utilization of contextual information as metadata for building and controlling service models. Though many technical and technological aspects of Web Service composition are abstracted away within the proposed framework, it represents a solid formal foundation for creation, deployment and execution control of Context-aware Composite Web Services.

5.2 Answers to Research Questions

We describe the results achieved in the thesis by answering the research questions, which have been formulated in Chapter 1.

The main research question set in the thesis has been formulated as follows:

How can context be utilized in the process of Web Service Composition to improve relevance, robustness and quality of composite Web Services?

The main research question has been generally answered by the design of the Context-aware Web Service Composition framework and appropriate specification of its component procedures, most of which make use of contextual information directly or indirectly to improve functioning of their analogues unaware of context. Below we give a more specific answer to the main research question.

We utilize context as an additional control facility for Web Service composition. More specifically, context helps better verify whether a service is suitable for certain purposes or not. What is more, context presents exclusive means to estimate whether a service is *currently* suitable to achieve particular goals or not. Thus, utilization of a context can significantly improve the quality of Web Services and their relevance to user-specific goals, since a context represents another dimension along which the correspondence between a goal and a service,

which achieves it, is tested. So, context can narrow the range of suitable services and make the match better. Apart from this, when we speak merely of composing services, context, being naturally a dynamic matter, can help shift component service selection from service creation phase to service execution phase where the choice between two similar services can be made exclusively based on instant contexts. Otherwise, if the choice is made prior to execution, it may be that the corresponding context changes meanwhile and the selected service will not fit to the changed context anymore, which will in turn drop the quality and relevance characteristics for the entire composite service. Therefore, with appropriate modeling of services, dynamicity of context can be used for the benefit of services' quality and relevancy. Moreover, such real-time usage of context can remarkably increase robustness of composite services because availability of component services can be treated as context. If a component service is selected prior to the execution of the composite service, there is a probability that this component service may encounter failure in the meantime and hence render a failure of the entire composite service. When component services are selected real-time, probabilities of their failures are negligibly small, since they are invoked almost immediately after their selection.

In order to realize some appealing perspectives opened by the described context utilization, we need the means to manipulate context(s) in a variety of ways, e.g., associating context(s) to Web Services, reasoning on context(s), etc. Although context modeling is basically out of the scope of this thesis, we have to explicitly state that such features of our solution as contextual preconditions of Web Services and context-based service selection are only possible to realize on the basis of Semantic Web Services. Modeling of contextual information using Semantic Web languages such as OWL and WSML to capture its explicit meaning and allow matchmaking procedures operating with captured semantics on top of rich ontological vocabularies is intuitively the only realistic way to reconcile Web Services and context.

This answer gives a general idea of how context(s) can be utilized in Web Service composition. To provide a more detailed analysis of such context utilization and to revise our results more deeply, we have to answer detailed research questions set in Chapter 1.

RQ 1: What are Web Services? How are they formally modeled and composed?

- We formally define Web Services as processes of sequential state changes. To be more rigorous, we can say that a Web Service can be defined as a partially-ordered set of states and operations. It is possible to say that a service always resides in a certain state or is always ascribed a state, and all possible states of the service are different. Operations are actions that switch the service from one

state to another. The service has an initial state and a final state and no operation can switch the service into its initial state and from its final state. Note that such definition is not exclusive to Web Services. Generic services, business processes, generic computational process or even some material processes can be modeled using this definition. This has its pros and cons. On one hand, such loose definition conforms to Web Services being a subclass of more general services, and it suits well to our modeling purposes. On the other hand, it does not consider the specificity of Web Services in comparison to more general types of services and processes.

- We formally model Web Service processes or, to be more technical, Web Service control flows using Petri nets modeling formalism. Petri nets ideally suit for process modeling (they were actually designed for that) by representing complex processes with two sets of elements, places and transitions, which together constitute a bipartite graph model via connecting pairs of elements from different sets with directed arcs. Places model states, while transitions model operations. Arcs basically express causal linkage with the flow. So, Petri net models present an intuitive and pictorially clear view of processes and particularly Web Services' control flows. To model Web Services we use ordinary Petri nets in a standard form to adequately capture the specifics of their control flow organization. Among the advantages of Petri net models is the availability of mature analysis and optimization techniques, including manipulation using declarative Petri net languages. As a relative drawback of Petri nets their poor scalability can be mentioned.
- Web Service process model is basically a workflow. Workflows can be generally split into control flows, data flows and resource flows [1, 2] according to the type of process they model. In this thesis we are mainly interested in the modeling of control flows. Complex control flows can be organized in different ways depending on what kind of logical and causal relationships process states exhibit. This relationships, e.g. concurrency, exclusiveness, sequence, etc. are realized using special workflow constructs called *patterns*. Workflows can be adequately and unambiguously modeled using Petri nets. Patterns are realizable in Petri net apparatus by appropriate constructs. The more expressive individual patterns are, and the more logically complete their collection is, the more powerful and precise the framework that we obtain for modeling Web Services.
- In Chapter 3 we presented a study of workflow patterns and investigated them with respect to their suitability for modeling Web Services.

- Petri net models of single Web Services can be combined into a composite Petri net model through abstracting single Petri net models as simple service operations in terms of the composite model. Then the process of composite service modeling is indistinguishable from the process of simple service modeling, provided that some of the service operations correspond to entire services. Due to their standard form layout, component Petri nets (we call them service nets) are combined using a unified set of composition patterns (operators/constructs). To model some of the composition constructs, which require zero condition testing, we however have extended our Petri net formalism by adding inhibitor arcs to it.
- In order to verify some vital functional properties of Web Services and ensure their flawless execution, standard Petri net analysis techniques, such as liveness check and reachability trees can be efficiently used. These techniques are simple and mature, and allow the verification of all Petri nets' properties, which are critical for Web Services' integrity, reliability and performability. Web Service composition process benefits from formal semantics modeled using Petri nets since the results of such a way of modeling can always be analyzed and verified using standard analysis tools, such as reachability trees.
- We developed a declarative Web Service algebra to perform algebraic manipulations with composite service structures modeled by Petri nets. Our algebra is unambiguously mapped to the algebra of Petri nets. It allows us to determine important algebraic properties of composite service structures and successfully utilize these properties to perform structural optimization of composite services' Petri net models.

RQ 2: What are context-aware Web Services? How are they formally defined?

- We define context-aware Web Services as Web Services capable of adjusting their execution flow using context information acquired at run-time. Adjustment of a control flow is made possible after the substitution of uncontrolled exclusion choices by controlled ones. This method allows initially non-deterministic branching of the control flow become deterministic. But actual determination of such branching is accomplished at the stage of control flow execution.
- Formally, we model context-aware Web Services using extended Petri net formalism called Meta Petri nets. Meta Petri net model of a context-aware Web Service consists of two separate Petri net structures. One is the service net of the identical Web Service, which

is unaware of context. It resides on the lower level of the model. The upper-level structure is the control net that models special context-aware functionality reconfiguring the control flow on the lower level.

- Modeling of context-aware Web Services differs from modeling of ordinary Web Services in three main aspects. Firstly, the Petri net model of a context-aware Web Service consists of two separate Petri nets executed concurrently. Secondly, the Petri net model of a context-aware Web Service may contain two or more (depending on the number of exclusion choices on the lower level) service nets, where each net represents a separate service composition. In contrast, a service net of an ordinary Web Service contains at most one service composition. Finally, the execution of the lower-level service net is controlled by special places on the upper level control net via reconfiguring the structure of the lower-level Petri net through dedicated meta-linkage between levels.

RQ 3: How can context utilization enhance Web Service composition process?

- Goal-driven composition planning procedure aims at creating service composition plans (control flow plans) from user-specific goals using AI planning techniques. In our solution we use the Hierarchical Task Networks planning technique, which allows decomposing user goals into subgoals and structuring these subgoals as goal trees. Context can be used to optimize structures of goal trees by removing some of their elements if corresponding subgoals do not suit into user context. To achieve that, we establish a sort of personalization functionality, where profiled user context is used to determine feasibility of goals and their subgoals.
- In our solution, context is also efficiently used for service matchmaking. More specifically, as soon as normal semantic matchmaking phase for component services is over, and component services are selected for inclusion into the built composition plan with respect to their correspondence to specific subgoals, context-based verification of services' suitability can be still applied to these candidate services in order to sharpen composition relevance and decrease the complexity of composition structure. We call this procedure service delegation. It is based on checking specific contextual preconditions of candidate services. These preconditions have to be defined similarly to normal preconditions and also provided as part of semantic service descriptions. Verification of these preconditions is the task of the semantic reasoner, which we do not consider in this thesis. Based on such verification some candidate

services, which have successfully qualified at the semantic matchmaking phase, can be still discarded from the composition due to their contextual inadequacy. However, if a contextual precondition engages such a context, which cannot be sensibly measured in advance and it cannot be ensured that this context will persist for the execution time of composition, the corresponding service should not be rejected from the composition. Instead, its context precondition will be checked during the execution phase. We feel that the context-aware service delegation procedure is a noticeable enhancement to traditional service matchmaking since it can guarantee that selected component services not only semantically fit to achieve user goals, but also will not fail in those goals' achievement with respect to the current context. This can significantly increase the quality of service compositions, while simultaneously decreasing their complexity.

- Scalability of Web Service compositions generally depends on three main factors: scalability of the composer, scalability of the modeling technique and the actual number of component services. Since we do not care about the composer (semantic reasoner) in this thesis, we do not take it into consideration when discussing scalability of our compositions. Petri nets as the modeling technique are not very scalable, which is a well-known deficiency of this formalism. However, we enhance the scalability of the method by introducing declarative service algebra for reduced notational and computational complexity. Finally, by utilizing context(s) for sorting out component services, we sometimes can remarkably reduce the number of component services and, hence, increase the scalability of the service composition.

RQ 4: How are context-aware Web Services composed?

- Composite context-aware Web Services are context-aware Web Services, which functionality can be explicitly modeled as the composition of the other Web Services. Composite context-aware Web Services reconfigure the structure of their service compositions with respect to contextual information acquired at run-time. The upper control layer of a composite context-aware Web Service contains one or more control nets. The number of these control nets corresponds to the number of controlled exclusion choices in the lower-level service net. Each control net represents a service net which models a composition of context provider services and contains some additional constructs that model contextual reasoning and non-functional reasoning functionality. As a result of execution of each control net one of two special metaplaces is marked. These

two places are associated with an exclusion choice in the lower-level service net. The marked place identifies which of the two alternatives is to be selected on the lower level. In this context-aware fashion the service net on the lower level is reconfigured at execution time dynamically determining the actual configuration of the Web Service's control flow.

- Semantically the choice between the alternatives on the service layer of a context-aware composite Web Service is based on the verification of the contextual preconditions of the component services that corresponds to the mentioned alternatives. Though contextual preconditions' verification is seen as an implicit composition reasoner's functionality, it is included into the model of a control net as an atomic operation, which assists in reconfiguring the structure of the main service net.
- Context-aware composite Web Service is a completely autonomous functional unit which is capable of run-time reconfiguration of its own structure based on the current context. Real-time context acquisition is performed by component context provider services belonging to the composite service. Such functions as context interpretation, reasoning, contextual preconditions' verification, non-functional reasoning and corrective action issuing are performed by the composition reasoner and can be also seen as component operations or component services within the structure of the composite service. In contrast, context-aware composition of Web Services refers to the process of composing ordinary Web Services (or context-aware Web Services) from component Web Services in a context-aware fashion. In particular this means that such crucial procedures as goal-driven planning and component service matchmaking are performed with regard to context. However, context-aware composition of Web Services does not itself mean that resulting composite Web Services are capable of reconfiguring themselves according to context, i.e., are context-aware. Thus, context-aware composition indicates that the process of service creation is context-aware, whereas composite context-aware Web Service refers to a service, the execution of which is context-aware.
- Meta Petri nets are the extended Petri net formalism which is particularly designed to model reconfigurable structures. It dictates hierarchical organization of Petri net models, in which the lower level is associated with a reconfigurable Petri net structure, while the upper layer models the control structure that performs actual reconfiguration of the structure on the lower level. This modeling mechanism perfectly suits our modeling purposes. The main

distinction of composite context-aware Web Service modeling compared to simple context-aware Web Service modeling is that the lower level of the model contains the actual composition of component Web Services, while in the case of simple context-aware Web Service its service net contains no composition. Nevertheless, since we showed before that in a composite service structure component services can be easily abstracted as simple service operations, there is no conceptual difference between modeling of composite and non-composite context-aware Web Services using Meta Petri nets. However, to meet our specific modeling requirements we extended the basic apparatus of Meta Petri nets with such advanced Petri nets elements as inhibitor arcs, disjunctive transitions, multiplexer transitions and our own invention – user adaptation transitions.

- Context-aware reconfiguration of service net structures implies the presence of a context reasoning functionality, which is supposed to perform matching of real-time acquired context to existing service requirements. Such contextual reasoning functionality is to be provided by the composition reasoner, as we already mentioned, and is included into the service as a set of operations representing remote procedure calls. In terms of context-aware Web Service structure contextual reasoning is implemented via a number of contextual precondition check procedures. Each context precondition check procedure represents a single call of composition reasoner's function, which matches a certain contextual variable with a single contextual precondition of a single component service within the composite Web Service's service net. This reasoner's function returns a binary answer which indicates whether the precondition is satisfied or not. In terms of Petri nets, we model this reasoning function as a special disjunctive transition. A component service is validated by the sets of contextual precondition check procedures, the number of which corresponds to the number of contextual preconditions posed by the service. The concrete Petri net organization of service validation procedure depends on the set of logical relationships among the contextual preconditions of a particular component service. The output of each service validation procedure can basically determine whether the corresponding component service has to be included into the service net or not. However, if more than one alternative component service is qualified based on the context, non-functional reasoning procedure must be performed to make a choice between the qualified alternatives. This type of reasoning procedure is performed similarly to the described above, but is modeled in a

slightly different way by special multiplexer transitions. Finally, if all alternative component services failed to qualify based on the context, user aware adaptation procedure, which is a specific type of reasoning procedure realizing the concept of passive context awareness, should be utilized. This procedure is again performed by the reasoner and is modeled by special user adaptation transitions in terms of Petri net model.

So, by giving these moderately detailed answers to the previously established research questions of the thesis, we reviewed the main research results obtained in the thesis and emphasized some important and innovative aspects of the proposed solution to Context-aware Web Service Composition problem.

In addition to the given description of the obtained results, below we specifically describe the major contributions of this thesis and then try to evaluate them by identifying the major benefits of the proposed solution.

5.3 Summary of Contributions

As the major contribution of this thesis, we developed the formal framework for context-aware composition of Semantic Web Services. Our framework utilizes context in several ways, namely, for the formulation of composition goals, for the description of Web Services and for the controlled execution of composite Web Services. Thus, it brings additional value and flexibility to the modern view on automated semantic Web Service composition process [58, 60].

The more specific contributions, which we would like to particularly emphasize in this thesis, and which represent more detailed constituents of the main contribution, are the following:

1. In Chapter 3 we investigated, in depth, the formal foundation of workflow-based Web Service composition, basic and advanced workflow composition patterns and their applicability to abstract scenarios. Based on the set of selected patterns we developed a declarative Web Service composition algebra, the operators of which are associated to available composition patterns and exhibit complex algebraic properties and relationships. Our algebra provides a convenient instrument for declarative manipulation with composite Web Service structures. It is fully compatible with the Petri net models of Web Services used and its properties allow for efficient structural optimization of those models.
2. In Chapter 4 we thoroughly studied the process of automated semantic Web Service composition and presented the generic view on it with respect to modern trends in the Web Service composition research field.

We specified our particular vision of this process on the basis of the described common view by proposing and comprehensively describing enhancements to its component procedures. Specifically, we proposed our particular approach to automatic modeling of composition goals, which among other features makes use of context information to refine goal models. We also contrived a context-aware enhancement to the procedures of component service matchmaking and composite service's control flow construction. The service delegation (as we called this enhancement) procedure allows selection of component Web Services with respect to currently observed context, which potentially leads to significant reduction of composite service's process model, potential increase of composition method's scalability, and better overall quality and relevance of value produced by built composite Web Services. Finally, we devised the mechanism of run-time reconfiguration of composite Web Services' control flow structures with respect to dynamic contexts. This mechanism relies on real-time verification of component services' contextual preconditions using specific context-aware functionality integrated into a composite service. Based on the results of such verification the mechanism makes a selection among multiple alternative control flow paths, which correspond to alternative component services.

3. We gave a process-oriented definition to the concept of context-aware Web Service. We introduced a Petri net-based formal method for modeling and composing context-aware Web Services. Using this method we formally specified the context-aware reconfiguration mechanism for composite context-aware Web Services. We showed how our goal-driven Web Service composition framework can be partially reused to build context-aware composite Web Services on top of usual composite Web Services.

5.4 Solution's Benefits

The main virtue of the solution to Web Service composition problem presented in this thesis is its *innovation*. Our solution brings in context as a new facet in describing and managing Semantic Web Services. Furthermore, it utilizes context to achieve a new degree of quality, flexibility and automation in Web Service composition.

More specific solution's properties, which we can classify as its benefits, are the following:

- *Formality*. Our solution is rather abstract. It merely exploits formal tools for description, analysis and composition of Web Services. This

leads to several important consequences. First of all, formal solutions are generally technology neutral. Indeed, we abstract from the underlying technological basis, which is necessary for our solution to be deployed. For example, we do not specify which Web Service orchestration engine should be used to run Web Services produced in conformance to our framework. Instead we claim that any available engine can be used if the formal operational semantics of our Web Services (expressed using Petri net model) can be unambiguously and correctly transcribed into the specification written in the orchestration language supported by this particular engine. Secondly, our solution does not dictate which particular knowledge representation languages should be used for semantically described Web Services. It is mediated from the specifics of such different representations by the semantic composition reasoner and can use any of them as long as such a reasoner is capable of interpreting and inferring knowledge on top of each particular representation. Finally, with some minor modifications our solution can be adopted to perform composition upon broader classes of services and processes.

- *High degree of personalization*. The proposed solution adds context as an extra dimension for describing Web Services. More specifically, context is used for describing the environment, in which Web Services are situated, and linking them to this environment, rather than explicitly describing Web Services themselves. Value-added composite Web Services are built within our framework in such way that their functionality can be customized with respect to not only user-specific goals, but also other contextual factors, which describe current state of a user and his/her environment or might not be related to the user at all. This solution provides a high degree of Web Services' personalization because it allows taking into account factors which are implicit from the viewpoint of interaction with the user.
- *Run-time composition*. Context-aware composite Web Services are built in such a way that their component Web Services are actually composed at the execution stage. At the planning stage component Web Services are not really composed. It is better to say that they are preselected. But not all of them will be included in the final control flow of the composite service. The choice between the preselected alternatives is made at execution time and on the base of current context by adjusting the structure of the composite service's control flow. This approach leads to the improved conformance of composed services to user-specific goals and requirements by instant matching of component services to dynamic environment in which interaction between the user and the service takes place. Another benefit is better

service quality attainable by real-time verification of QoS parameters. Finally, composite service reliablility is increased since component services' failure by absence is significantly less probable in the case of their late integration into composite service's control flow structure.

- *Gradual reactivity*. Reactivity is basically seen as an ability of computational entity to (instantly) react to the changes in certain external factor(s). Even this loose definition makes it clear that reactivity is a direct consequence of active context awareness. We specify the core reconfiguration mechanism of our Context-aware Web Service Composition framework using primarily the principles of active context awareness, so this mechanism does exhibit reactivity. We can say that our reconfiguration procedure is reactive because it adjusts the structure of composite service's control flow in response to the instant values of the associated contextual variables measured immediately before the actual reconfiguration of the particular part of the control flow. We call this gradual reactivity because all single parts of the control flow, which are subjected to context-aware reconfiguration, are basically reconfigured independently of each other and one after another in the order determined by the control flow. Reactivity is to be perceived as a significant enhancement to Web Service composition process, since it makes composite services generally more flexible and responsive. Final composition of context-aware composite Web Service from component Web Services services is reactive because it is made at the composite service's execution time, which is hardly possible in frameworks that compose services prior to the execution phase.
- *Proactivity*. Although we stated above that our solution is reactive, it is also a proactive one. Proactivity in our framework is incarnated by the capability of forcing certain actions to change certain external factors in critical situations instead of waiting for appropriate values of these factors to appear and then reacting to them. In cases where our reactive approach to context-aware reconfiguration of composite Web Service's control flow fails to validate any of the available alternative component services in the observed context, the execution process should either wait until context changes so that at least one alternative can qualify in it, or take a proactive approach and force the user to change context deliberately or revise his/her requirements to the service being composed. This latter way of dealing with such a critical situation is more efficient in terms of composition robustness and execution time. However, its relative drawback is in possible distraction of the user from other activities.

There are no perfect solutions. Our solution is no exception. However, its relative drawbacks and deficiencies, which we are able to identify so far, are rather unaddressed and still unresolved issues, which remain out of scope, due to time and level of detail restrictions applied to the thesis. We will further discuss these issues in the section devoted to future work.

5.5 Directions for Future Work

We identify two major directions for the continuation of our work. Since the Context-aware Web Service Composition framework presented in this thesis is a completely theoretical solution lacking implementation details, one apparent direction of this solution's refinement would be its practical implementation, at least for demonstrative purposes. The other way of polishing the developed solution is through increasing efficiency, flexibility and scalability of the formal framework. Below we discuss some of the possible enhancements in greater detail.

5.5.1 Practical Implementation

Practical implementation (even for demonstrative purposes) has been kept out of the scope of this thesis because development of even a rather simple application framework, which would realize our formal framework on top of the existing technologies of Web Services and the Semantic Web, would pose some serious technological challenges, a successful solution of which would be worth a separate thesis. So, the road to practical implementation of the Context-aware Web Service Composition framework, should it ever happen, will have been paved with significant contributions which solve the variety of the mentioned challenges. Here we identify some of the most important contributions that must be made to apply our framework to its full potential:

- *Mapping of Petri net service models into Web Service orchestration language*. In order to produce executable composite Web Services, their Petri net process models should be properly and unambiguously convertible into the markup of at least one Web Service orchestration language, such as BPEL4WS or WS-CDL. Then composite Web Services can be executed by a dedicated Web Service orchestration engine. Though such conversion procedure may seem plain and simple, the problem might be quite serious because Meta Petri nets, which we are using for modeling of context-aware composite Web Services, are not generally convertible into constructs of known Web Service orchestration languages. In the best case necessary mapping can be easily found, whereas in the worst scenario

establishing of the mapping will require modifications to both our formal modeling method and existing orchestration languages.

- *Specification of semantic composition reasoner*. When we speak about different aspects of our formal framework's applicability to solve specific Web Service composition challenges and problems throughout Chapter 4, we often rely on the notion of semantic composition reasoner as the central decision-making facility implementing our formalism. Apparently, in terms of our framework's practical application this reasoner must be thoroughly specified before being implemented or adopted. Specification of semantic composition reasoner is non-trivial task. First, the language for internal knowledge representation and reasoning on it should be chosen. This reasoner and its internal language should support operational semantics expressed using our Web Service algebra and/or Petri nets (Meta Petri nets) formalism. In addition, it must pledge comprehensive support to Web Services' native capabilities (SOAP/WSDL/UDDI) and various Semantic Web languages to be capable of reasoning based on standard semantic service descriptions and ontologies. With the extensive support of the underlying Web Services standards and with an appropriate design the composition reasoner would be able to compose all semantically described Web Services and even expose its own reasoning functions in the form of Web Services to be included into the control functionality of context-aware Web Services.
- *Context modeling using semantic ontology languages*. To be able to perform reasoning based on context, the composition reasoner has to understand what a particular context means, i.e., context must be described semantically. Modeling of context is also a non-trivial task. The first question to be answered is how should contextual information be modeled to enable contextual reasoning for Web Service composition? A variety of formal approaches exist. A proper approach for context modeling should be carefully chosen by evaluating some important characteristics of different approaches as described in [106]. Although any approach can be used, if its context models are supported by the composition reasoner, we are sure that ontological modeling of context is the most suitable choice, because in addition to functional merits it provides greater interoperability and openness of context-enabled Web due to unified technological basis for all semantic representatitions of information and knowledge on the Web. Another crucial question is more specific. How can contextual preconditions and effects be represented using standard semantic service description languages? Or otherwise how to modify semantic service description standards to accommodate contextual preconditions and effects as part of Web Services' descriptions? Approaches like RgbDF [54] and C-OWL [16] can be used as take-off on

the way to implementation of context-aware semantic Web Service composition and context-enabled Web in general.

- *Automated composition goal description*. User-specific composition goals and composition goals generated automatically by agents or inside Web Service composition frameworks should be semantically described to enhance reasoning capabilities of composition reasoners. Although we discussed some aspects of automated goal description in Chapter 4 and developed some ideas as to how to appropriately describe composition goals for derivation of goal-driven composition plans, ideally the approach to goal description should become more open and standard. Introduction of semantic goal specification language and design of goal ontologies for automatic interpretation and reasoning on top of such language's goal descriptions would be a universal and powerful solution for the entire Web.

5.5.2 Enhancements to Formal Method

The formal framework for context-aware Web service composition, which we developed in this thesis, represents comprehensive solution approach. However, some of its aspects were not worked out to the smallest details or can be enhanced in more significant ways. Here we identify some of the most valuable and promising ideas aspiring to increase the quality and the power of our formal approach to Web Service composition.

- *Precomposition*. Precomposition mechanism should be specified and added to our formal framework. The main idea of precomposition is the following. Whenever we have two or more alternative component services, one of them must be finally selected for execution and the others discarded. This is made via gradual assessment of component services' relevance, preconditions and effects, contextual preconditions and effects, and finally QoS parameters. Given appropriate descriptions, which formally specify all these service properties or at least provide a possibility for their assessment, component services are rather easily qualified for inclusion into a composite service and compared. But this simplicity no longer holds if we have to choose among sequences of component Web Services. In contrast to single component Web Services (which, by the way, can also consist of sequences of other component services), sequences of Web Services do not have Web Service descriptions associated to them, since they are not exposed as individual Web Services that can be reused in other composite Web Services. No service description automatically means impossibility of assessment and comparison with other alternative

component services. Precomposition procedure aims to solve precisely this type of problem. It proposes to perform virtual service composition process and as a result obtain description of the virtual service composed as the given sequence of component services. Virtual service and virtual composition here mean that component services are composed merely formally, "on paper", but not in practice, i.e., such virtual composite service will never be exposed and consumed as a valid Web Service, it will never reach the execution stage and never function in the form of a real application. However, it can be reused in other Web Service composition processes if the obtained virtual service description is logged and stored somewhere. Precomposition procedure would substantially increase the power of our service composition framework because it provides means for creating more flexible structures of composite services' control flows. This procedure would be especially useful for determining QoS characteristics of service sequences and performing efficient QoS service composition, where alternative services are actually compared to each other based on their QoS properties.

- *QoS composition*. Although we included QoS composition procedure into our composition framework and even modeled it as part of control functionality within composite context-aware Web Services, it is be better to admit that we only reserved a place for QoS composition within our approach. Next step is to study and specify criteria for QoS-based service comparison, which is now considered as an implicit part of the composition reasoner. In addition to explicit modeling of QoS composition procedure it should be made more independent in the sense that in certain circumstances Web Services should be composed merely with respect to their QoS characteristics. In the current view of our framework this is impossible and QoS composition plays only an auxiliary role in context-aware composition by refining the results of its operation. Furthermore, explicit consideration of QoS may help in determining how QoS parameters of context provider services within control layer of a context-aware composite Web Service influence its QoS characterstics. This type of assessment in combination with the precomposition procedure can be utilized to make a more technically grounded decision: whether context should be used during a composite Web Service's execution or not.
- *Data flow modeling*. When Web Services are combined into composite structures, the order of their invocation and execution is determined by control flows of composite services. Modeling of control flows is generally the substance of the work done in this thesis. However, control flow is not the only flow linking component services. Since services have inputs and outputs, they can exchange data as well. Normally messages transferred among Web Services facilitate execution of both control flow and data

flow. Data are usually sent from service to service along with the corresponding control influence. But in some cases data can be sent by one service to another independently of control influences. These cases can make difference for the execution of composite services by presenting more complex relationships among component services than those stipulated by the corresponding control flow structure. So, data flows should be also explicitly modeled to bring more flexibility to our service composition framework.

- *Compensation mechanism*. In the current version of our service composition framework no procedure for execution backtracking is provided. This is a serious limitation because Web Services are generally unreliable and can fail or crash during their execution. That is why the ability to roll back their execution to the initial state is a crucial feature to be included in the framework.
- *Simulation tool development*. We use the formal method of Meta Petri nets to model composite Web Services structures. In the thesis we do not provide any evaluation of the method. Nevertheless, using special simulation software the method can be assessed numerically and compared to other possible modeling formalisms. Should such simulation tool be developed, it would not only help evaluate the method, but also would allow us to understand the qualities and deficiencies of the selected formalism and would possibly give us ideas how to refine it to improve functional characteristics of the specified Web Service composition process.

5.6 Concluding Remarks

In this Chapter we provided a brief but capacious overview of the results produced in the thesis. We gave concise answers to the research questions established in Chapter 1. We surveyed the most important contributions we made in this thesis. We identified the main benefits of the Context-aware Web Service Composition framework, which is the major contribution of this thesis. Finally we described our vision of possible future work, directions applied to both practical implementation of the framework, and the refinement of the formal methodology underlying the framework.

Our principal conclusion is that the Context-aware Web Service Composition framework is an innovative solution to Web Service composition problem, and can open new horizons in quality, relevance and flexibility for Web Service composition process. However, there are many important questions to be answered and many serious issues to be tackled before context-aware Web Service composition becomes a reality.

REFERENCES

[1] W.M.P. van der Aalst, and A.H.M. ter Hofstede, *Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages*, in Proceedings of The Fourth International Workshop on Practical Use of Coloured Petri Nets and the CPN Tools, 2002, pp. 1-20.

[2] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros, *Workflow Patterns*, Distributed and Parallel Databases, vol. 14(3), July 2003, pp. 5-51.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, *Business Process Execution Language for Web Services (BPEL4WS) Version 1.1*, Technical Report, BEA Systems, Microsoft, SAP AG and Siebel Systems, 2003, http://ifr.sap.com/bpel4ws/BPEL%20V1-1%20May%205%202003%20Final.pdf (last retrieved 13.12.2006).

[4] A. Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. McDermott, S.A. McIlraith, S. Narayanan, M. Paolucci, T Payne, and K Sycara, *DAML-S: Web Service Description for the Semantic Web*, Proceedings of the 1st International Semantic Web Conference (ISWC), Springer Verlag, 2002, pp. 348-363.

[5] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek, *Web Service Choreography Interface (WSCI) 1.0*, W3C Note, August 2002, http://www.w3.org/TR/wsci/ (last retrieved 13.12.2006).

[6] A. Arkin, "Business Process Modeling Language (BPML)", November 2002. http://www.bpmi.org/.

[7] A. Alves, A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C.K. Liu, D. König, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu, "Web Services Business Process Execution Language 2.0", OASIS TC Specification, May 2006, http://www.oasis-open.org/apps/org/workgroup/wsbpel/ (last retrieved 13.12.06).

[8] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H. Yang, *The Case for Cyber Foraging*, in Proceedings of the 10th ACM SIGOPS European Workshop: Beyond the PC, ACM Press, 2002, pp. 87-92.

[9] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski, *Challenges: An Application Model for Pervasive Computing*, in Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking, 2000, pp. 266-274.

[10] L. Barkhuus, and A.Dey, *Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Examined*, in Proceedings of Ubicomp 2003: Ubiquitous Computing 5th International Conference, 2003, pp. 149-156.

[11] B. Benatallah, Q. Z. Sheng, and M. Dumas, *The Self-Serv Environment for Web Services Composition*, IEEE Internet Computing, vol. 7(1), January-February 2003, IEEE Educational Activities Department, pp. 40-48.

[12] M. Benerecetti, P. Bouquet, and C. Ghidini, *Contextual Reasoning Distilled*, Journal of Experimental and Theoretical Artificial Intelligence (JETAI), 12(3), 2000, pp. 279-305.

[13] F. Bennett, T. Richardson, and A. Harter, *Teleporting – Making Applications Mobile*, in Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, 1994.

[14] T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic Web", Scientific American, Vol. 284(5), May 2001, pp. 34-44.

[15] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, *Web Service Architecture,* W3C Working Group Note, February 2004, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, (last retrieved on 13.12.2006).

[16] P. Bouquet, F. Giunchiglia, F. Van Harmelen, L. Serafini, and H. Stuckenschmidt, *Contextualizing Ontologies*, Journal of Web Semantics, vol. 1(4), 2004, pp. 1-19.

[17] J.M. Bradshaw, *An Introduction to Software Agents*, in Software Agents, Bradshaw, J.M. (ed.), Cambridge, MA: MIT Press, 1997.

[18] T. Bray, D. Hollander, A. Layman, and R. Tobin, *Namespaces in XML 1.0 (Second Edition)*, W3C Recommendation, August 2006, http://www.w3.org/TR/REC-xml-names/, (last retrieved on 6.12.2006).

[19] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan, *Extensible Markup Language (XML) 1.1 (Second Edition)*, W3C Recommendation, August 2006, http://www.w3.org/TR/2006/REC-xml11-20060816/, (last retrieved 6.12.2006).

[20] J. de Brujin, H. Lausen, A. Polleres, and D. Fensel, *The Web Service Modeling Language WSML: An Overview*, in Proceedings of the 3rd European Semantic Web Conference (ESWC 2006), Springer, LNCS 4011, 2006.

[21] F.Buschmann, R.Meunier, H.Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture—A System of Patterns*, Wiley, NY, 1996.

[22] M. Carman, L. Serafini, and P. Traverso, *Web Service Composition as Planning*, in Proceedings of ICAPS 2003 Workshop on Planning for Web Services, 2003.

[23] G. Chen, and D. Kotz, *A Survey of Context-Aware Mobile Computing Research* Dartmouth Computer Science Technical Report TR2000-381, 2000.

[24] H. Chen, T. Finin, and A. Joshi, *An Ontology for Context-Aware Pervasive Computing Environments*, ACM Knowledge Engineering Review, Vol. 18(3), 2003, pp. 197-207.

[25] H.Chen, T. Finin, A. Joshi, L. Kagal, F. Perich, and D. Chakraborty, *Intelligent Agents Meet the Semantic Web in Smart Spaces*, IEEE Internet Computing, vol. 8(6), IEEE Educational Activities Department, 2004, pp. 69-79.

[26] E. Christinsen, F. Curbera, G. Meredith, and S. Weerawarana, *Web Service Definition Language (WSDL) Version 1.1*, W3C Note, March 2001. http://www.w3.org/TR/wsdl, (last retrieved on 6.12.2006).

[27] L. Clement, A. Hately, C. von Riegen, and T. Rogers, *UDDI Version 3.0.2 Specification*, OASIS TC Specification, February 2005. http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm, (last retrieved 12.12.2006).

[28] D. Connolly, F.van Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein, *DAML+OIL Reference Description*, W3C Note, December 2001, http://www.w3.org/TR/daml+oil-reference, (last retrieved on 6.12.2006).

[29] G. Couloris, J. Dollimore, T. Kindberg, *Distributed Systems Concepts and Design (Third Edition)*, Addison-Wesley, 2001.

[30] A. J. Demers, *Research Issues in Ubiquitous Computing*, in Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, ACM Press, 1994, pp. 2-8.

[31] J. Desel and G. Juhas, *What Is a Petri Net? - Informal Answers for the Informed Reader*, in Unifying Petri Nets: Advances in Petri Nets, Hartmut Ehrig et al. (Eds.), LNCS 2128, 2001, pp. 1-25.

[32] A.K. Dey, and G.D. Abowd, *Towards a Better Understanding of Context and Context-Awareness*, in Proceedings of the Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems, 2000.

[33] A.K. Dey, *Understanding and Using Context*, Personal and Ubiquitous Computing, vol. 5(1), Springer-Verlag, pp. 4-7.

[34] A.K. Dey, D. Salber, and G.D. Abowd, *A Conceptual Framework and Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*, Human-Computer Interaction, vol. 16(2-4), 2001, pp. 97-166.

[35] M. Ebling, G. Hunt, and H. Lei, *Issues for Context Services for Pervasive Computing*, in Proceedings Advanced Topic Workshop on Middleware for Mobile Computing in association with IFIP/ACM Middleware 2001 Conference, 2001.

[36] V. Ermolayev, N.Keberle, O. Kononenko, S. Plaksin, and V. Terziyan, *Towards a Framework for Agent-Enabled Semantic Web Service Composition*, International Journal of Web Services Research, 1(3), 2004, pp. 63-87.

[37] M. Esler, J. Hightower, T. Anderson, and G. Borriello, *Next Century Challenges: Data-Centric Networking for Invisible Computing: The Portolano Project at the University of Washington*, in Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking (Mobicom 99),ACM Press, 1999, 256-262.

[38] D.C. Fallside, and P. Walmsley, *XML Schema Part 0/1/2: Primer, Structures, Datatypes (Second Edition)*, W3C Recommendation, October 2004, http://www.w3.org/TR/xmlschema-0/, (last retrieved on 6.12.2006).

[39] D. Fensel, and C. Bussler, *The Web Service Modeling Framework WSMF*, White Paper, 2002, http://www.swsi.org/resources/wsmf-paper.pdf, (last retrieved on 10.12.2006).

[40] P. Flajolet, and C. Puech, *Partial Match Retrieval of Multidimensional Data*, Journal of ACM, vol. 33(2), 1986, pp. 371-407.

[41] J. Flinn, D. Narayanan, and M. Satyanarayanan, *Self-Tuned Remote Execution for Pervasive Computing*, Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, IEEE Computer Sociaety, 2001, p. 61.

[42] J. Floch, S. Hallsteinsen, A. Lie, and H.I. Myrhaug, *A Reference Model for Context-Aware Mobile Services*, in Proceedings of NIK 2001, 2001.

[43] M. Fluegge, I.J.G.Santos, N.P. Tizzo, and E.R.M. Madeira, *Challenges and Techniques on the Road to Dynamically Compose Web Services*, in Proceedings of the 6th international conference on Web engineering, ACM Press, 2006, pp. 40-47.

[44] E. Gamma, R.Helm, R.Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Mass., 1994.

[45] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, *Project Aura: Towards Distraction-Free Pervasive Computing*, IEEE Pervasive Computing, vol 1(2), IEEE Educational Activities Department, 2002, pp. 22-31.

[46] T.R. Gruber, *A Translation Approach to Portable Ontologies*, Knowledge Acquisition, vol. 5(2), 1993, pp. 199-220.

[47] T. Gu, H.K. Pung, and D.Q. Zhang, *Toward an OSGi-Based Infrastructure for Context-Aware Applications*, IEEE Pervasive Computing, vol. 3(4), 2004, pp. 66-74.

[48] H. Haas, A. Brown, *Web Services Glossary*, W3C Working Group Note, February 2004, http://www.w3.org/TR/ws-gloss/, (last retrieved on 8.12.2006).

[49] R. Hamadi and B. Benatallah, *A Petri Net-Based Model for Web Service Composition*, in Proceedings of 14th Australian Database Conference, 2003, pp. 191-200.

[50] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, *The Anatomy of a Context-Aware Application*, Wireless Networks, vol. 8(2-3), 2002, pp.187-197.

[51] J. Hendler, *Agents and the Semantic Web*, IEEE Intelligent Systems, vol. 16(2), IEEE Educational Activities Department, 2001. pp. 30-37.

[52] K. Henricksen, J. Indulska, and A.Rakotonirainy *Modeling Context Information in Pervasive Computing Systems*, , in Proceedings of 1st International Conference on Pervasive Computing, Springer-Verlag, LNCS 2414, 2002, pp. 167-180.

[53] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto, *Web Service Choreography Description Language (WS-CDL) Version 1.0*, W3C Candiadate Recommendation, November 2005, http://www.w3.org/TR/ws-cdl-10/, (last retrieved on 13.12.2006).

[54] O. Kaykova, O. Khriyenko, V. Terziyan, and A. Zharko, *RgbDF: Resource Goal and Behaviour Description Framework*, in Proceedings of the 1st IFIP WG12.5 Working Conference on Industrial Applications of Semantic Web, 2005, pp. 83-99.

[55] J. Keeney, and V. Cahill, *Chisel: A Policy-Driven, Context-Aware, Dynamic Adaptation Framework*, in Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, IEEE Computer Society, 2003, p. 3.

[56] M. Keidl, and A. Kemper, *Towards Context-Aware Adaptable Web Services*, in Proceedings of the 13th International World Wide Web Conference, 2004, pp. 55-65.

[57] J. Koehler, and B. Srivastava, *Web Service Composition - Current Solutions and Open Problems*, in Proceedings of the ICAPS 2003 Workshop on Planning for Web Services, 2003, pp. 28 - 35.

[58] D. Kuropka, and H. Meyer, *Survey on Service Composition*, Technical Report of the Hasso-Plattner-Institute, 10 (2005), ISBN 3-937786-78-3, ISSN 1613-5652,

http://kuropka.net/files/HPI_10_Serv-Comp-Survey.pdf, (last retrieved on 13.12.2006).

[59] F. Leymann, "Web Services Flow Language (WSFL 1.0)", IBM, 2001.

[60] M. Lin, H. Guo, and J. Yin, *Goal Description Language for Semantic Web Service Automatic Composition*, in Proceedings of The 2005 Symposium on Applications and the Internet (SAINT'05), vol. 00, IEEE Computer Society Press, 2005, pp. 190-196.

[61] O. Lassila, *Serendipitous Interoperability*, in Proccedings of the Semantic Web Kick-Off in Finland – Vision, Technologies, Research, and Applications, HIIT Publications 2002-001, University of Helsinki, 2002.

[62] O. Lassila, and M. Adler, *Semantic Gadgets: Device and Information Interoperability*, in "Ubiquitous Computing Environment", Case Western Reserve University, Kalle Lyytinen & Yongjin Yoo (eds.), 2003.

[63] O. Lassila, and D. Khushraj, *Contextualizing Applications via Semantic Middleware*, in Proceedings of The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous), IEEE Computer Society, 2005.

[64] R. Lehtonen, and J. Harju, *Access Network Independent Service Control System for Stream Based Services*, in Proceedings of the EUNICE 2000 Summer school, Enschede, Netherlands, pp. 23-30.

[65] C. M. MacKenzie, K. Laskey, F. McCabe, P.F. Brown, and R. Metz, *OASIS Reference Model for Service Oriented Architecture V 1.0*, OASIS Official Committee Specification, approved August 2006, http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf, (last retrieved on 6.12.2006).

[66] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, , S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, *OWL-S: Semantic Markup for Web Services*, W3C Member Submission, November 2004, http://www.w3.org/Submission/OWL-S/, (last retrieved on 6.12.2006).

[67] B. McBride, *RDF Primer*, W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-primer/, (last retrieved on 6.12.2006).

[68] B. McBride, *RDF Vocabulary Description Language 1.0: RDF Schema*, W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-schema/, (last retrieved on 6.12.2006).

[69] J. McCarthy, *Notes on Formalizing Context*, in Proceedings of the 13th International Joint Conference on Artificial Intelligence, 1994, pp. 555-560.

[70] D.L. McGuinness, and F. van Harmelen, *OWL Web Ontology Language Overview*, W3C Recommendation, February 2004, http://www.w3.org/TR/owl-features/, (last retrieved on 6.12.2006).

[71] S. McIlraith, T.C. Son, and H. Zeng, *Semantic Web Services*, IEEE Intelligent Systems, vol. 16(2), IEEE Educational Activities Department, 2001, pp. 46-53.

[72] B. Medjahed, A. Bouguettaya, and A.K. Elmagarmid, *Composing Web Services on the Semantic Web*, The VLDB Journal, vol. 12(4), Springer-Verlag, 2003, pp. 333-351.

[73] Merriam-Webster's Dictionary, http://www.merriam-webster.com/dictionary, (last visited 7.12.2006).

[74] H. Meyer, and D. Kuropka, *Requirements for Web Service Composition*, Technical Report of the Hasso-Plattner-Institute, 11 (2005), ISBN 3-937786-81-3, http://kuropka.net/files/HPI_11_Serv-Comp-Req.pdf, (last retrieved 13.12.2006).

[75] Microsoft, *.NET Framework*, www.microsoft.com/net/.

[76] N. Milanovic, and M. Malek, *Current Solutions for Web Service Composition*, IEEE Internet Computing, vol. 8(6), IEEE Educational Activities Department, 2004, pp. 51-59.

[77] R. Milner, *The Polyadic π-Calculus: A Tutorial*, in Logic and Algebra of Specification, Springer-Verlag, 1993, pp. 203-246.

[78] N. Mitra., *SOAP Version 1.2 Part 0/1/2: Primer, Messaging Framework, Adjuncts*, W3C Recommendation, June 2003, http://www.w3.org/TR/soap12-part0/, (last retrieved on 6.12.2006).

[79] T. Murata, *Petri Nets: Properties, Analysis and Applications*, in Proceedings of the IEEE, 77(4), 1989, pp. 541-580.

[80] S. Narayanan, and S. McIlraith, *Simulation, Verification and Automated Composition of Web Services*, in Proceedings of the 11th International World Wide Web Conference, 2002, pp. 77-88.

[81] D. Nickull, *Service Oriented Architecture*, White Paper, Adobe Systems, Inc., May 2005.

[82] OASIS, *OASIS ebXML Collaboration Protocol Profile and Agreement (CPPA) v.2.0*, http://www.oasis-open.org/committees/download.php/204/ebcpp-2.0.pdf, (last retrieved on 6.12.2006).

[83] OASIS, *OASIS ebXML Registry TC*, http://docs.oasis-open.org/regrep/v3.0/regrep-3.0-os.zip, (last retrieved on 6.12.2006).

[84] M.P. Papazoglou, and D. Georgakopoulos, *Service-Oriented Computing*, Communications of the ACM, vol. 46(10), ACM Press, 2003, pp- 25-28.

[85] M.P. Papazoglou, *Service-Oriented Computing: Concepts, Characteristics and Directions*, in Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03), 2003, pp. 3-12.

[86] J. Pascoe, *Adding Generic Contextual Capabilities to Wearable Computers*, in Proceedings of the Second International Symposium on Wearable Computers, IEEE Computer Society Press, 1998.

[87] C. Perkins, *Mobile IP*, IEEE Communications Magazine, vol. 35(5), 1997, pp. 84-99.

[88] C. Perkins, *Mobile Networking Through Mobile IP*, IEEE Internet Computing, vol. 2(1), IEEE Educational Activities Department, 1998, pp.58-69.

[89] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc., 1981.

[90] S.R. Ponnenkanti, and A. Fox, *SWORD: A Developer Toolkit for Web Service Composition*, in Proceedings of the 11th World Wide Web Conference, 2002.

[91] K. Raatikainen, *Functionality Needed in Middleware for Future Mobile Computing Platforms*, in Proceedings of ACM Advanced Topic Workshop on Middleware for Mobile Computing in association with IFIP/ACM Middleware 2001 Conference, 2001.

[92] J. Rao and X. Su, *Toward the Composition of Semantic Web Services*, in Proceedings of GCC 2003, LNCS 3033, Springer-Verlag, 2004, pp. 760-767.

[93] J. Rao, and X. Su, *A Survey of Automated Web Service Composition Methods*, in Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, July 2004.

[94] W. Reisig, Petri Nets: An Introduction, EATCS Monographs on Theoretical Computer Science, vol. 4, Springer-Verlag, 1985.

[95] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, *Web Service Modeling Ontology*, Applied Ontology, 1(1), pp. 77 - 106, 2005.

[96] D. Saha, and A. Mukherjee, *Pervasive Computing: A Paradigm for the 21st Century*, IEEE Computer, vol. 36(3), IEEE Computer Society Press, 2003, pp. 25-31.

[97] M. Satyanarayanan, *Fundamental Challenges in Mobile Computing*, in Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, ACM Press, 1996, pp. 1-7.

[98] M. Satyanarayanan, *Pervasive Computing: Vision and Challenges*, IEEE Personal Communications, August 2001, pp.10-17.

[99] V. Savolainen and V. Terziyan, *Metapetrinets for Controlling Complex and Dynamic Processes*, Information and Management Sciences, vol. 10(1), 1999, pp. 13-32.

[100] B. Schilit, N. Adams, and R. Want, *Context-Aware Computing Applications*, in Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, 1994, pp. 85-90.

[101] D.C. Schmidt, *The ADAPTIVE Communication Environment: An Object-Oriented Network Programming Toolkit for Developing Communication Software*, Concurrency: Practice and Experience, vol. 5(4), 1993, pp. 269-286.

[102] D.C. Schmidt, *Using design patterns to develop reusable object-oriented communication software*, Special issue on object-oriented experiences and future trends, Communications of the ACM, vol. 38(10), 1995, pp. 65-74.

[103] N. Shadbolt, T. Berners-Lee, and W. Hall, *The Semantic Web Revisited*, IEEE Intelligent Systems, vol. 21(3), IEEE Educational Activities Department, 2006, pp. 96-101.

[104] E. Sirin, J. Hendler, and B. Parsia, *Semi-automatic Composition of Web Services using Semantic Descriptions*, in Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS 2003, 2002.

[105] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, *HTN Planning for Web Service Composition Using SHOP2*, Journal of Web Semantics, 1(4), 2004, pp. 377-396.

[106] T. Strang, and C. Linnhoff-Popien, *A Context Modeling Survey*, in Proceedings of UbiComp 2004 1st International Workshop on Advanced Context Modelling, Reasoning and Management, 2004, pp. 34-41.

[107] Sun Microsystems, *Java 2 Platform, Enterprise Edition (J2EE)*, http://java.sun.com/j2ee/.

[108] S. Thatte, XLANG*: Web Services for Business Process Design*, Microsoft, 2001, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, (last retrieved on 13.12.2006).

[109] A. Tolk, and J.A. Muguira, *The Levels of Conceptual Interoperability Model (LCIM)*, in Proceedings of IEEE Fall Simulation Interoperability Workshop, IEEE Computer Society Press, 2003.

[110] A. Toninelli, R. Montanari, L. Kagal, and O. Lassila, *A Semantic Context-Aware Access Control Framework for Securing Collaborations in Pervasive Computing Environments*, in Proceedings of the 5th International Semantic Web Conference (ISWC 2006), Isabel Cruz et al (eds.), Springer Verlag, Athens (GA), November 2006.

[111] A. Tsalgatidou and T. Pilioura, *An Overview of Standards and Related Technology in Web Services*, Distributed and Parallel Databases, vol. 12(2-3), 2002, pp. 135-162.

[112] M. Vukovic, and P. Robinson, *Adaptive, Planning Based, Web Service Composition for Context Awareness*, International Conference on Pervasive Computing, 2004.

[113] M. Weiser, *The Computer for the 21st Century*, Scientific American, September 1991, pp. 94-104.

[114] M. Weiser, *Some Computer Science Issues in Ubiquitous Computing*, Communications of the ACM, vol. 36(7), 1993, pp. 75-84.

[115] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia, *Automatic Web Services Composition Using SHOP2*, In Proceedings of Workshop on Planning for Web Services in conjunction with ICAPS 2003, June 2003.

[116] S.S. Yau, F. Karim, Y. Wang, B. Wang, and K.S. Gupta, *Reconfigurable Context-Sensitive Middleware for Pervasive Computing*, IEEE Pervasive Computing, vol. 1(3), IEEE Computer Society, 2002, pp. 33-40.

[117] C. Yushi, Lee. E. Wah, and D.K. Limbu, *Web Services Composition – An Overview of Standards*, ITSC, Synthesis Journal 2004, October 2004, pp. 137-150.

[118] D. Zhovtobryukh, *Integration Issues in Communication Environments*, in Proceedings of the 1st International Workshop on Ubiquitous Computing IWUC 2004, 2004, pp. 28-37.

[119] D. Zhovtobryukh, and N. Kohvakko, *Service Reference Model for Modern Communications*, in Proceedings of the Fourth International Network Conference INC2004, S.M. Furnell and P.S. Dowland (eds.), 2004, pp. 221-228.

[120] D. Zhovtobryukh, and V. Hara, *Service Portability Framework for Integrated Communication Environments*, in Proceedings of the 1st IFIP WG12.5 Working Conference on Industrial Applications of Semantic Web, Springer-Verlag, V.Terziyan and M. Bramer (eds.), 2005, pp. 317-340.

[121] H. Zimmermann, *OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection*, IEEE Transactions on Communications, vol. 28(4), 1980, pp. 425 - 432.

YHTEENVETO (FINNISH SUMMARY)

Verkkopalvelun muodostaminen (web service composition) on laajasti tunnettu ongelma verkkopalveluiden tutkimuksessa sekä teollisuudessa että akateemisissa piireissä. Sen väitetään olevan hyvin merkittävä tulevaisuuden kehityksessä ja nykyisten hajautettujen laskentajärjestelmien ja teknologioiden evoluutiossa, erityisesti WWW-ympäristöissä. Ottaen huomioon nykypäivän merkittävät tekniset ja menetelmälliset vaatimukset, ongelman ratketessa mahdollistuisi sähköisten palvelujen laadun, käytettävyyden ja joustavuuden kasvu, josta hyötyisi sekä palvelun tarjoajat että palvelun käyttäjät.

Tämä väitöskirja on omistettu tarkalleen tämäntyyppisen kattavan ja innovatiivisen ratkaisun luomiseen verkkopalveluiden muodostamisongelmaan. Erityisesti väitöskirja keskittyy ongelmanratkaisun menetelmälliseen näkökulmaan yrittäen rakentaa yleisen tavan palveluiden muodostamiseen sitomatta ratkaisua pelkästään tiettyyn konkreettiseen joukkoon väliohjelmistoja ja tiedonesitysstandardeja. Pikemminkin ratkaisu noudattaa avoimuuden periaatteita ja teknologista neutraalisuutta, mutta silti perustuu tiukkoihin analyyseihin nykyisistä käytännöllisistä ja teoreettisista tuotoksista ja niiden tehokkaaseen uusiokäyttöön.

Väitöskirjatyö on palveluorientoituneen laskennan (service oriented computing), semanttisen verkon (semantic web) ja kontekstitietoisen laskennan (context aware computing) paradigmojen risteyskohdassa. Työn tavoitteet ovat yhteneväisiä suurimman osan nykyisen palvelun muodostamisen tutkimusaktiviteettien ja laajasti hyväksyttyjen ja oleellisten tavoitteiden kanssa. Näiden tavoitteiden joukossa ovat seuraavat vaativat haasteet:

- Automatisoitu palvelun muodostaminen
- Tavoiteohjattu palvelun muodostaminen
- Dynaamisesti tehty palvelun muodostaminen

Vaikkakin osaan näistä haasteista on jo saatu hyviä menetelmällisiä ratkaisuja, tämä väitöskirjatyö pyrkii osittain uudistamaan nykyisiä ratkaisuja ja osittain käyttämään uutta innovatiivista lähestymistapaa.

Esitetyn muodostamistavan innovatiivisuus pitäisi nähdä ensisijaisesti kontekstitietoisen koostetun verkkopalvelun esilletuonnissa. Konteksti nähdään uutena verkkopalvelun kuvauksen julkisivuna ja uutena ulottuvuutena verkkopalvelun muodostamisen hallinnassa. Lisäämällä kontekstitietoisuus webbipalveluihin ja verkkopalveluiden muodostamiseen käytettäviin kehysrakenteisiin ei pelkästään nosteta koostettujen palveluiden laatua, käytettävyyttä ja joustavuutta vaan myös tehostetaan muodostusmekanismien toiminnallisuutta parantamalla koostettujen palveluiden merkityksellisyyttä, vakautta ja skaalautuvuutta. Eräs esimerkki erittäin innovatiivisesta tuotoksesta,

joka löytyy väitöskirjasta on kontekstitietoinen dynaaminen uudelleenkonfigurointitoimenpide verkkopalveluille. Tämä tarkoittaa erityisesti mukana tulevan kontekstitietoisen toiminnallisuuden suunnittelua, joka voidaan laittaa suoraan mukaan palveluun myöhempää palvelurakenteen uudelleenkonfigurointia varten suorituksen aikana.

Kehitty menetelmä perustuu kypsään formaaliin mallinnustyökaluun – Petri-verkkoihin. Tämä formalismi sisältää joitakin merkittäviä ominaisuuksia, jotka antavat syyn käyttää sitä palvelun muodostamisen mallinnustyökaluna. Erityisesti sen käytettävyys palvelun muodostamiseen voidaan tehokkaasti todistaa. Se tarjoaa tehokkaita ominaisuuksia tilasiirtymäjärjestelmien mallinnukseen (joita verkkopalvelut käytännössä ovat), se tarjoaa korkean tason formalismin ja laajan, mutta lyhytsanaisen notaation ja lisäksi se tarjoaa itseanalysoitavat ja tarkistettavat mallit. Menetelmä lähtee liikkeelle yksinkertaisista Petri-verkkojen piirteistä esittäen työnkulkuketjuja (workflows) ja päättyy hienostuneeseen Petri-verkkojen erikoistyyppiin, jota kutsutaan Meta Petri -verkoiksi esittäen koko kirjon kontekstitietoisia koostettuja verkkopalveluita.

Väitöskirja on jaoteltu niin, että se tarjoaa riittävän loogisen, hyvin järjestetyn, yksityiskohtaisen ja eheän esityksen kontekstitietoisen verkkopalvelun muodostamisen kehysrakenteesta ja pakollisista työvaiheista. Väitöskirja alkaa johdannolla ja työn motivoinnilla, sen jälkeen jatkaa tarvittavien taustatietojen ja nykytilan kuvaamisella ja lopulta siirtyy päätuotosten esittelyyn: verkkopalveluiden formaaliin mallintamiseen Petri-verkkojen avulla ja kontekstin käyttämiseen koostettujen verkkopalveluiden muodostamisessa kontekstitietoisesti sekä kontekstitietoisten koostettujen verkkopalveluiden luomiseen.

Avainsanat: verkkopalvelu, palvelun muodostaminen, semanttinen verkkopalvelu, konteksti, kontekstitietoisuus, automatisointi, uudelleenkonfiguroitavuus, Petri-verkot.