

This is a self-archived version of an original article. This version may differ from the original in pagination and typographic details.

Author(s): Tuunanen, Tuure; Przybilski, Michael

Title: Domain Specific Case Tool for ICT-Enabled Service Design

Year: 2014

Version: Published version

Copyright: © 2014 IEEE

Rights: In Copyright

Rights url: <http://rightsstatements.org/page/InC/1.0/?language=en>

Please cite the original version:

Tuunanen, T., & Przybilski, M. (2014). Domain specific case tool for ICT-enabled service design. In Proceedings of the 47th Hawaii International Conference on System Sciences (HICSS 2014) (pp. 955-964). IEEE. Proceedings of the Annual Hawaii International Conference on System Sciences. <https://doi.org/10.1109/HICSS.2014.126>

Domain Specific Case Tool for ICT-Enabled Service Design

Tuure Tuunanen
University of Jyväskylä
tuure@tuunanen.fi

Michael Przybiski
University of Helsinki
michael.przybisk@cs.helsinki.fi

Abstract

One major problem in service design is the limited availability of information gathered during the development process. In particular, information on end-user requirements is difficult for designers, developers, and maintainers to access. Here, we provide a mechanism that supports the gathering and modeling of various types of information throughout the service and software development life cycle. As various existing tools focus on a particular part of the life cycle, essential information is not available, or it is more difficult to obtain in later stages. The linkage between information collected in the different stages is often lost. The implemented tool support enables the modeling of requirements; the abstraction of these requirements in the form of the required system functionalities, which can also be modeled; and the connection with component-based software engineering to support the design of ICT-enabled services.

1. Introduction

Information and communications technology (ICT)-enabled services are the new frontier for information systems research. Tuunanen et al. [1] has defined ICT-enabled services as “systems that enable value co-creation through the development and implementation of information and communication technology enabled processes that integrate system value propositions with customer value drivers.” As such, they are accustomed to seeking entertainment or even pleasure through different kinds of ICT-enabled services. These services go beyond the well-known Web-based and digital services, which have become the focus with smartphone and smart television apps, as well as different social media services. The technological applications of such services will include, for example, hardware-based sensors and real-time data analytics.

The focal point is that the infusion of ICT to services and the service-dominant logic thinking [2] in ICT development is a game changer. This calls attention to interesting problems that have not yet

been fully studied. More specifically, we see that the infusion of ICT into the services creates similar problems that have been already previously faced with software and information systems development. One of these challenges is how we can maintain the connection between users’ needs and goals for the service and the requirements information that analysts and developers use to design the service.

During a typical software and information systems development process, large amounts of information are collected, such as end-user requirements from interviews, software design artifacts, pseudo code, or test cases. This information is typically collected and provided in a more abstract form to the next stage, while the connection to the originating material is often not maintained any further. In our previous research [3-5], we have provided a formal approach for the conversion of verbal and content rich requirements [5, 6] into systems requirements. This enabled us to maintain the connection between information that is gathered during the various stages of the service and software life cycle,¹ and to give, for example, service and software designers and developers easy access to the gathered requirements information. However, the tool support for linking the requirements and the development life cycle was not available in the literature.

This paper reports the tool development, which empowers us to easily model the various requirements in the form of their “features,” “outcomes,” and “values” [6]. The tool, additionally, provides us with the means to bridge the gap between analysts and developers [7], by enabling the modeling of service functionalities and software components, as well as the relationships between these entities. Thus, our objective is to facilitate co-creation activities namely between analysts and developers. We utilized design science research [8] as our research methodology. Hevner et al. [8] posited that design science research can be used to develop constructs, models, methods, and instantiations. Our research develops an instantiation of a domain specific case tool for ICT-enabled service design.

¹ After this, we use “development life cycle” for this.

Furthermore, to enable a large number of developers to take advantage of the modeled information, we implemented our tool using the Java programming language and integrated it with the popular and widely used Eclipse² IDE (integrated development environment). Eclipse has the advantage of being available for a large number of operating systems and platforms, as well as of supporting most major programming languages. As it is extremely flexible, Eclipse can be used not only for software development, but also for any kind of project. Eclipse is also the foundation of several other development platforms, such as the Carbide IDE,³ used for developing software for mobile phones. This allows our tool to be used for requirements modeling in all kinds of projects, for various programming languages and platforms, and to reach a large number of people involved in different stages of software development projects.

The remainder of this article is structured as follows. In the next section, we provide an overview of requirements modeling literature, as well as model-driven development (MDD). In the next section, we describe the details of the developed tool, how it can be used to model user requirements, and how it supports our approach to modeling ICT-enabled service requirements and bridging the gap between analysts, designers, and software developers, based on component-based software engineering [7, 9]. Thereafter, we describe how our tool supports the development of ICT-enabled services, based on the modeled requirements. We conclude this paper by describing our further research and necessary work on the subject.

2. Background

As our approach is based on providing a connection between requirements elicitation and more technical ICT-enabled service design, we next provide a brief overview of the state-of-the-art requirements modeling and model-driven development.

2.1 Requirements Modeling

One of the first steps in the development life cycle is the elicitation of end-user requirements. Academics, as well as practitioners, are constantly trying to find better ways to elicit, analyze, and model requirements. Current trends are moving toward richer requirements and more complex, multi-dimensional requirements information [5, 10]. This allows for a better description of the collected requirements and a better understanding of the

intended meaning of complex and ambiguous answers from end-users.

A prevailing problem is the insufficient availability of the collected requirements and the derived information during the later stages of the development life cycle. Studies by various authors, such as [11], have shown how to successfully convey the needs of end users to analysts, but the current literature offers no straightforward solutions on how to extend the communication to designers and developers when using advanced requirements engineering (RE) methods. Moreover, the current, more designer-oriented requirement engineering methods, such as scenarios [12], often start with a different agenda for requirements elicitation. Instead of trying to determine the requirements, they frequently rely on contextual factors of a use-situation [13] or a scenario of a probable use-situation. Similarly, another well-used method, prototyping, usually assumes that it is already possible to present something to the potential users, such as a mock-up or a prototype of the application.

Earlier, Peffers and Tuunanen [5] have demonstrated the efficient use of rich requirements information in combination with laddering and theme clustering to elicit end-user requirements and provide a link between end-users and analysts. In order to implement the gathered requirements, it is necessary to find a means of expressing them in such a way that a computer system can interpret them correctly. Recent developments in this direction have led to the increased use of models that can automatically or with adjustments by developers be converted into a binary format for a specific platform.

2.2 Model Driven Development

Finding the best level of abstraction for a particular problem domain is one of the biggest problems in software development [14]. By definition, MDD attempts to provide a level of abstraction that focuses on modeling the behavior of software entities. As a result, the outcomes of the MDD approach are not computer programs, but models [15, 16]. The advantage of this approach is the possibility of developing and expressing concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain, which makes the models easier to specify, understand, and maintain. Improving the process of understanding needs and requirements, as well as how they map to software by using models, significantly reduces the risks that come with the development and implementation of complex solutions and allows us to find solutions more easily.

² <http://www.eclipse.org>

³ <http://www.forum.nokia.com>

Providing a higher level of abstraction is, however, not always sufficient or desirable. Another advantage of MDD is the possibility of providing specifics that we are familiar with and that are required for specific problem solutions. Thus, through the use of models, software also becomes more understandable, as understandability is a direct function of the expressiveness of the used modeling form [7, 16].

In MDD, a model attempts to provide all relevant representations to the real-life system and the problem. Thus, it is also possible to find an accurate solution, as the result is produced from the model using automated tools. This also ensures that, based on the same model, the result will always be exactly the same. Using an MDD approach, it is furthermore possible to correctly predict the interesting (perhaps even missing), relevant, but less obvious characteristics of the modeled system. This can be achieved by further experimentation—for instance, by executing the model in a computer or by formal analysis.

While it is still expensive to obtain an accurate model, the derivation of the computer program is done automatically. Because supporting programs performs the actual implementation for a specific platform, the MDD approach is significantly less elaborate than other approaches. MDD also provides the relatively simple adaptation to changes in the underlying infrastructure and allows the verification of the developed model at a very early stage of the development process, thus avoiding unnecessary and expensive modifications or even partial re-implementations.

As can be observed from the history of compiler technology, providing a higher level of abstraction often comes with an initially higher overhead and poorer efficiency. As the tools evolve, however, and become more sophisticated, the overhead is being continually reduced. In the same way that current compilers are significantly more efficient than the average human developer, when using a lower-level programming language, MDD tools also have the potential to provide more efficient code than humanly possible.

Because the outcomes of MDD projects are a set of models, rather than computer programs, they are completely independent of a programming language and its constrictions and limitations [7, 17]. Furthermore, assuming that the same basic characteristics apply, or provided that the appropriate support functionality is available, the development of software based on models is completely independent of underlying hardware architectures or middleware functionalities. It also provides complete

independence of the evolution of the model itself, from the changes in underlying systems.

A key factor in MDD is, however, that programs are generated automatically from their corresponding models [18]. Using models merely to document the developed software not only defies the primary idea of MDD, but it poses the danger that the models will not be maintained, particularly as the software evolves and adapts to new requirements.

In his paper [19], Ambler distinguishes two current Model Driven Development approaches. “Generative MDD,” being very much related to the Object Management Group,⁴ is a very idealistic approach, concentrating on sophisticated tools that allow for the development of advanced and sophisticated models, which can then be automatically transformed into software for particular platforms. Furthermore, it allows exchanges with other languages and formats, such as the Unified Modeling Language (UML) and the UML profiles or the specification language for model transformations [20].

While generative MDD is heavily based on the existence of sophisticated tools—first for the development of models and later on for the translation into specific code [7, 21]—“agile MDD” is based on the idea that modeling is a way of motivating thinking and consideration that is necessary before the actual implementation, as well as the tools that support it. Both approaches, however, take the use of tools into account and are even, to a greater or lesser extent, dependent on them.

In our previous research [3], we have provided an approach to transforming collected rich user requirements to system requirements that designers and developers can better understand and use. We are now providing an editor, consisting of a set of tools that together provide integration with existing modeling techniques, such as the unified approach, using UML and component-based software engineering—see, e.g., [22]. This integration provides a connection between advanced RE methods and current design practices, as well as a stronger link between the analyst and the designer, without limiting the possibilities of developing radical and innovative solutions. Furthermore, the approach enables us to retain the already gained link between end users and analysts. Additionally, it allows us to integrate designers and developers, with their knowledge of the target domain, as well as their expertise and experience, into the information cycle.

⁴ <http://www.omg.org/>

3. Research Methodology

Our paper explores the new service potential of mobile presence technology in connection with a research program that was under the auspices of a larger research project of LTT Research, Inc., a commercial research firm owned by the Helsinki School of Economics. The research program included 15 researchers from four continents and some 450 field study participants in Auckland, Helsinki, Hong Kong, and Las Vegas.

The presence technology allows mobile device users to share information about their current availability and status in terms of their own concepts or those of a presence-based application with subscribers to that information. For example, a basic presence service could allow users to publish their information and share it with others in order to make mobile communication and services more sensitive and personal. This information may include the availability of the subscriber, the preferred means of communication, the subscriber's whereabouts, as well as visual content for self-expression of one's emotion, in order to guide other users' communication decisions while controlling their own information [23]. Examples of presence information might include "sleeping," "in a meeting—leave voicemail," "bored—call me," or "at leisure and looking for fun."

Our scientific approach employs design science research [8]. Design science research complements both qualitative and quantitative research methodologies by using the development and design of artifacts to assist in the formulation of theories. According to Hevner et al. [8], the artifacts can be constructs, models, methods, and instantiations. Peffers et al. [24] extend this notion and reflect upon the thoughts of van Aken [25] and add that artifacts could also include social innovations or, as Järvinen [26] stated, new properties of technical, social, and/or informational resources. More recently, the topic of what exactly constitutes a design science theory [27] has been debated, which will be discussed after elaborating upon the foundation of our research methodology.

Design science research methodology (DSRM) [24] suggests a way to conduct design science research in information systems. It is comprised of six phases: (1) identify the problem and motivation; (2) define the objectives; (3) design; (4) demonstrate; (5) evaluate; and (6) communicate [24]. The DSRM starts with the identification of research problem(s) and the motivation for the research. Based on evidence, reasoning, and inference, the process continues toward defining the objectives of a solution to solve the research problem. This process should be

based upon prior knowledge in the given field of research. This knowledge is then used to design and develop an artifact and to create "how-to" knowledge. Following that, the artifact is used to solve the pre-described problem. Thus, it is demonstrated in a suitable context before evaluating its effectiveness or efficiency. This approach leads to disciplinary knowledge, which is then communicated to both academia and industry. Of course, the process can, and should be, iterative in nature. DSRM has four possible entry points to the research process. The first entry point is the traditional problem-centered initiation, which is similar to qualitative and quantitative research methodologies. The second is the objective-centered solution approach, which enables researchers to approach the research endeavor by first setting objectives that can be quantitative or qualitative in order to establish how the new artifact is expected to support solutions that achieve the stated objectives. The third entry point is design-centered, in which initiation can be a result of an interesting design or development problem. The fourth entry point is where the design starts with a research client. Our study takes a solution-based approach and develops an instantiation of a domain-specific case tool that addresses the problem of maintaining the connection between requirements information and designers and developers.

For requirements data collection and analysis, we applied a version of the critical success chains method [6] as part of the field study. It is a user-centered requirements development method for discovering and analyzing requirements data based on user preferences and reasoning for system applications and attributes from across user groups. In the following, we describe the general critical success chains method and its adaption used in this study.

First, the critical success chains method starts with an in-depth interviewing technique to elicit and discover user requirements. This technique is called laddering. The use of this interviewing technique does not require that participants have prior knowledge of the system, firm, or technology [6]. The laddering technique [28] is based on the personal construct theory [29] and has been widely used in marketing research [30, 31]. The output of the laddering interviews are chains of information in the format of "feature – reasoning – value or goal." These chains depict not only the feature-related requirements of the user, but also the reasoning behind these and possible goals or values that drive the user's behavior.

Second, as part of the interview process, the interviewees were asked to reflect on their ideas and also assign a numeric score to indicate how important

these system ideas were to them. The researchers then carry out a thematic cluster analysis to identify distinct themes and later group chains from the gathered data into the identified themes. For each theme, an interpretive clustering analysis process is carried out to further aggregate different expressions for similar ideas, consequences, and values.

Third, to provide a graphical representation of all chains within a theme, network maps were generated for each theme. The data collection and its analysis has been report more in detail in [4]. This research focuses on reporting how the critical success chains method can be integrated into model driven development and more specifically to domain-specific modeling. For this purpose, we have developed a custom modeling case tool. In the next section, we depict the general parts of the tool and thereafter the actual developed artifact.

4. Modeling

In this section, we first describe the general parts of the editor that enable us to model the different aspects of the diagram related to requirements, functionalities, and software components. Based on an example taken from a case study on mobile presence services [3, 4], we then describe the different parts of the diagram and their modeling functionalities, in turn.

4.1 Editor

Figure 1 shows the palette of the editor containing the various modeling tools. The three tools in the first section of the palette are used for the general manipulation of the diagram.

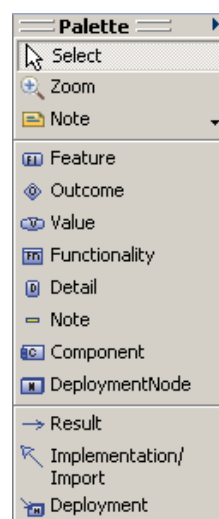


Figure 1. The Modeling Tools of the Editor

The “Select” tool allows for the selection of a specific object in the diagram. The second tool allows the user to zoom out of the diagram in order to get a better overview or to go to a specific section of the diagram. The user can also zoom in, in order to view more details and data. The “Note” tool enables the user to add free-text notes to any part of the diagram. These notes can also be connected to specific objects in the diagram. In case the connected object is moved to another part of the diagram, the connection will be maintained.

The next section of the palette contains the different nodes of the diagram that can be created. The first three tools are used to model the “Features,” “Outcomes,” and “Values” of a network diagram. A connection between these notes can be created with the “Result” relationship tool, which is found in the third section of the palette and described later.

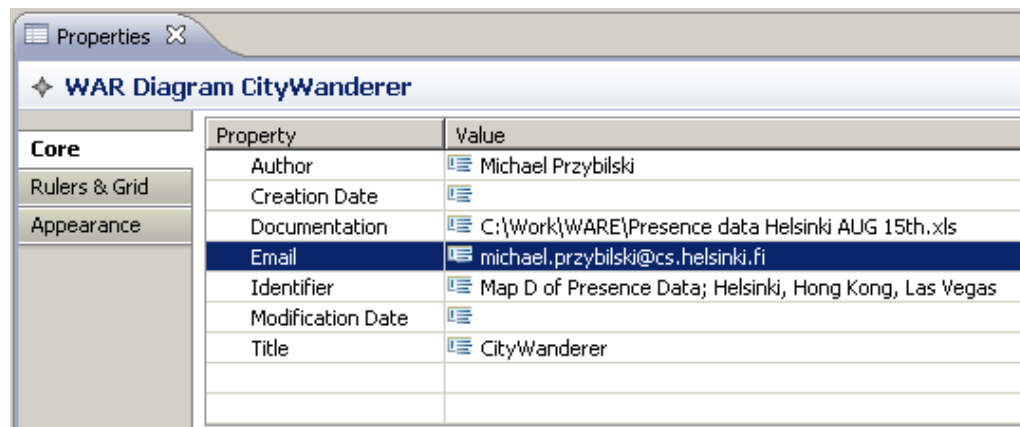
The next three tools in the palette allow the modeling of functionalities that can be derived from features, outcomes, and values. It is possible to add specific details to these functionalities, as well as notes that can contain information that has been provided during the interview process or in the first part of the requirements modeling. With the “Implementation/Import” tool, it is possible to link specific features, outcomes, or values to general functionalities or specific details.

The last two tools in the second section provide the connection to component-based software development. Using the “Component” tool, it is possible to model a component that will implement a particular functionality or a detail thereof. The connection between functionalities or their details and a component can be modeled using the “Implementation/Import” tool. This tool also allows the modeling of import relationships between different components. In this way, a component that consists of several sub-components can easily be modeled. The tool “DeploymentNode” allows the description of physical nodes on which the components will be deployed. The deployment of a component on a node can be specified using the “Deployment” tool in the last section of the palette.

The last section of the palette contains the different mechanisms that allow the modeling of connections between the various diagram objects. As briefly described above, the “Result” tool allows us to model how a feature results in one or several outcome(s), which, in turn, can result in one or several values. The “Implementation/Import” tool enables the modeling of the relationships between different features, outcomes, and values, as well as their resulting functionalities. It is also used to model how different functionalities are implemented by specific components and how components may

import other components. The last item in the palette, the deployment tool, enables the modeling of the components that are deployed on specific hardware nodes.

The editor also provides the analyst with a simple mechanism for specifying general properties of the model that is being developed, such as a title, the author's name and e-mail address, and the creation and last modification dates. Further information includes an identifier of the diagram, as well as further documentation of the modeled case or a link to it. These properties are shown in Figure 2.



Property	Value
Author	Michael Przybiski
Creation Date	
Documentation	C:\Work\WARE\Presence data Helsinki AUG 15th.xls
Email	michael.przybiski@cs.helsinki.fi
Identifier	Map D of Presence Data; Helsinki, Hong Kong, Las Vegas
Modification Date	
Title	CityWanderer

Figure 2. Properties of the Diagram

Based on the case study on mobile presence services [3, 4], we now describe the specific modeling elements of the editor as they are used during the typical development life cycle. We start with the modeling of requirements; as they are collected in the requirements gathering process, we show how the derived functionalities are created and how the components implement these functionalities.

4.2 Requirements

Following the critical success chains method process, after the interviews with the lead users, it is necessary to structure the collected requirements. For this purpose, we distinguish between “Features,”⁵ “Consequences,” and “Values.”⁶ Each stimulus that was provided to the lead user in the interview is typically modeled in a separate diagram and reflected in the diagram's title or documentation. Figure 3 shows an overview of a partial requirements map,

titled “City Wanderer,” which was part of one case study.

Visible on the left side of the diagram—in the form of blue, rounded rectangles—are some of the various features that were provided in the interview and their relationships. As can be seen, different features may result in one common feature, while one feature may result in different, more detailed features.

The provided features, in turn, result in different outcomes, modeled in the form of white trapezoids. These relationships between features and outcomes, as well as the relationships between different

outcomes, can also be modeled in the diagram.

The underlying values, as seen by the interviewed lead user, can also be modeled in the same way. Values are visualized in the form of yellow ellipses, while the relationships between outcomes and values, as well between the different values, are also modeled as described before.

This way, the tool provides the analyst with a simple and consistent mechanism for modeling the various aspects of the gathered requirements. Furthermore, each feature, outcome, or value also has a score, describing how important the user considered a specific feature. This information can also be modeled in the diagram. Furthermore, it is possible to maintain a link to the initial source of the information by providing the source chain or chains, as well as a reference to the initial interview (e.g., a link to the audio-file recorded during the interview). This information provides a link throughout the development process and makes it possible to further examine the source of a particular requirement and related details that might be unclear—e.g., due to misinterpretation or insufficient modeling.

⁵ “Attributes” in the original critical success chains methodology

⁶ “Values/Goals” in the original critical success chains methodology

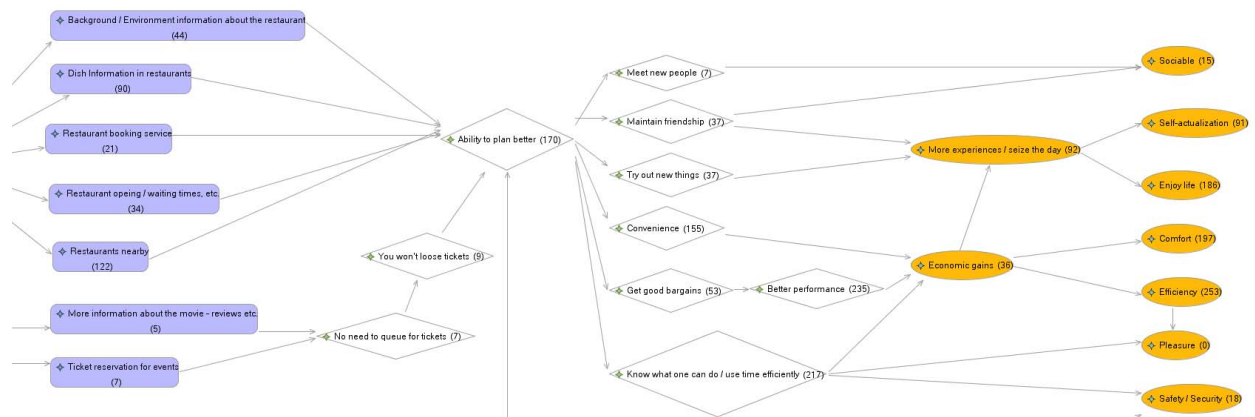


Figure 3. Part of a Requirements Map (actual screen shot of the tool)

4.3 Functionalities

In order to bridge the gap between the analyst and the developer, we decided that it was necessary to specify the separate system functionalities that would be part of a particular user requirement or a set thereof. These functionalities would serve as an intermediary between the modeled requirements and the software components that would implement them.

Our tool allows for the specification of functionalities on two different levels. On a more coarse level, it is possible to specify the functionalities themselves and link them to the features, outcomes, or results or to the user requirements that they satisfy. On a more fine-grained level, it is possible to specify the details of those coarse functionalities, which can also be related to specific requirements. Furthermore, these details can contain specific notes, enabling the analyst to describe, for example, variations of a specific detail coming from the interview(s).

Two functionalities that have been derived from the City Wanderer requirements map and their details are depicted in Figure 4. Using the tools, described earlier in Section 4.1, it is easily possible to model the different functionalities and their relationship to the requirements. Meta-information, such as the scoring of the different functionalities, is also maintained, based on the connection to their originating requirements. This information can serve as important data in the decision-making process—for example, to decide on the functionalities to be implemented or the order of their implementation.

4.4 Components

Functionalities allow us to bridge the gap between user requirements and the software that satisfies these requirements. On the software engineering side, it is possible to use various engineering methodologies, such as object-oriented software development. We decided to use a component-based software engineering approach and provide a connection to component diagrams. Figure 5 illustrates the components that implement the previously modeled “Information Provisioning” and the “Map” functionalities (see Figure 4).

The meta-data from other parts of the diagram are also maintained in this stage of the modeling process. Additional information related to the components—such as their platform, originating library, or implementation sources—can also be provided, and this information is used in the code generation process. Furthermore, it is possible to model the components that import other components and provide a complete application prototype.

In addition, one or several deployment nodes can be modeled. These nodes can specify details—e.g., of the hardware on which the components will be deployed as described in Figure 5.

5. Domain-Specific Case Tool for ICT-Enabled Service Design

Also visible in Figure 5 is the context menu of the map view component. This menu contains a cascading sub-menu, which allows the start of code generation for either the selected component or the selected component and all related components (i.e.,

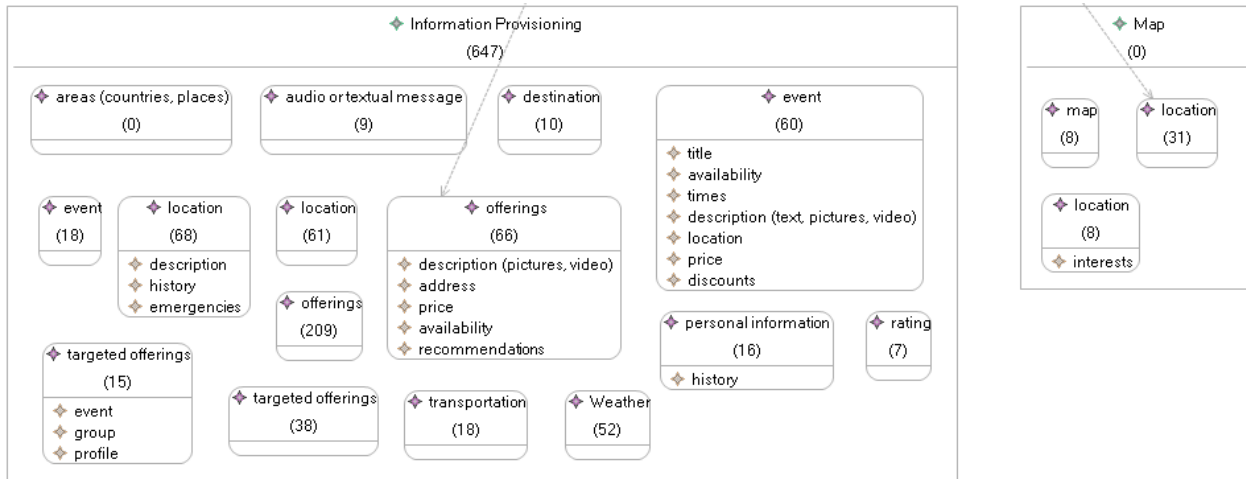


Figure 4. “Information Provisioning” and “Map” Functionalities

connected to the component and its deployment node). The information on the selected component and, if chosen, related components, is evaluated and, together with details of the deployment node, is used to generate code for the target platform.

To generate code, we currently use Java Emitter Template (JET) technology, which provides us a simple means of specifying how the modeled information should be combined with source code for different platforms. Depending on the target platform, it may then be necessary to set up the build environment, in order to build the generated code, which can then be run.

While the previously described method is focused on the elicitation of requirements from wide audience end users and the derivation of component models that satisfy these requirements, other requirements for engineering and development methods may, under other circumstances, be more suitable. Our editor was, thus, implemented in such a way that it could easily be integrated with other tools that support these engineering methods.

5.1 Integration

The modeled information is stored in two separate files, which are both in XML format. All information regarding requirements, functionalities, or components, as well as their relationships, are kept in one file, while all graphical information (e.g., regarding their layout in the diagram) is kept in a separate file. While any tools that are also integrated in the Eclipse IDE can use the provided API, other tools can use the XML files to access the modeled information and import the data into their own format.

5.2 Separate Diagrams

In order to enable the structuring into different diagrams, we provide a mechanism that allows the linking of entities in one diagram in another diagram. In this way, it is possible to not only re-use information that has already been modeled in one diagram in another, but also, the information is updated automatically. Changes in one diagram can automatically reflect in the related diagrams.

6. Conclusions

Based on the methodology developed in our previous research [3-5], we now provide a tool that enables continuous linkages between the various stages of the service development process, especially in service design. Based on rich requirements collected from end-users, we have shown how these user requirements can be modeled, how they can be abstracted in the form of functionalities, and how they result in system requirements. These can, in turn, be modeled directly in the form of software components, without losing the connection to previously collected information. Because this tool is integrated with the widely used Eclipse IDE, designers and developers have easy access to data that was used to derive the original requirements for their work. Thus, we see that our study contributes to the model-driven development and domain-specific modeling literature by offering a tool that integrates to a popular open source IDE. Some earlier domain-specific modeling tools were focused more on tailored software packages, such as MetaEdit+ available from MetaCase, Ltd.—see, e.g., [32].

Based on the gathered information, we were also able to demonstrate the generation of the prototype

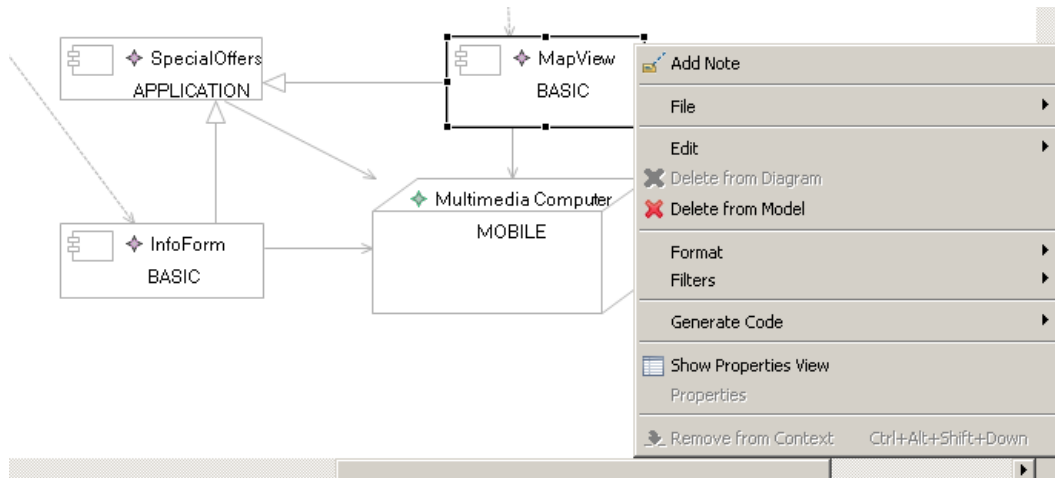


Figure 5. Components that implement the Functionalities, and the Service Application, as well as their Deployment Node

code, which can be compiled and run in an emulator or directly on a target device. This enables analysts to directly model the requirements and generate a functional prototype, allowing them to evaluate the collected requirements. Requirements and the derived functionalities can also be used to determine which functionalities are of particular importance for the end user and, thus, the order in which they should be implemented. This way, the gathered information can be used to design and develop a variety of ICT-enabled services, depending on the user's needs. Because connections to all gathered and modeled information are maintained throughout the development life cycle, it can also be used in later iterations. Furthermore, the tool was designed and developed in such a way that it can be easily integrated with other tools and used with other requirements and software engineering techniques. This ensures a high level of flexibility and also enables other researchers to evaluate their methodologies.

Our further research in this area focuses on three areas. Firstly, we are working on the further integration of our tool with other requirements in the engineering and service development methods. This will help in the refinement and the use of our tool in other fields of research. Secondly, we feel that the use of state-of-the-art code generation would provide a considerably higher impact of the developed tool and the refined methodology. Our tool is currently limited to Java code and the generation of basic prototype services, even though these applications

can be built and deployed immediately. Finally, we should further focus on the evaluation of the tool in various ICT-enabled service projects. While we have used several case studies to develop and evaluate our tool, a further evaluation in other fields and further case studies will be beneficial, as would be a gradual adoption in the industry.

References

- [1] T. Tuunanen, M. Myers, and H. Cassab, "A Conceptual Framework for Consumer Information Systems Development," *Pacific Asia Journal of the Association for Information Systems*, vol. 2, p. 5, 2010.
- [2] S. L. Vargo and R. F. Lusch, "Evolving to a New Dominant Logic for Marketing," *Journal of Marketing*, vol. 68, pp. 1-17, 2004.
- [3] M. Przybilski and T. Tuunanen, "From rich user requirements to system requirements," in *Proceedings of 11th Pacific-Asia Conference on Information Systems, Auckland, New Zealand, 2007*, pp. 561-573.
- [4] T. Tuunanen, K. Peffers, C. Gengler, W. Hui, and V. Virtanen, "Developing Feature Sets for Geographically Diverse External End Users: A Call for Value-based Preference Modeling," *JITTA : Journal of Information Technology Theory & Application*, vol. 8, pp. 41-55, 2006.
- [5] K. Peffers and T. Tuunanen, "Planning for IS applications: a practical, information theoretical method and case study in mobile financial services," *Information & Management*, vol. 42, pp. 483-511, 2005.

- [6] K. Peffers, C. Gengler, and T. Tuunanen, "Extending Critical Success Factors Methodology to Facilitate Broadly Participative Information Systems Planning," *Journal of Management Information Systems*, vol. 20, pp. 51-85, 2003.
- [7] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*, Minneapolis, MN, 2007, pp. 37-54.
- [8] A. R. Hevner, S. T. March, and J. Park, "Design Research in Information Systems Research," *MIS Quarterly*, vol. 28, pp. 75-105, 2004.
- [9] J. Grudin, "Interactive Systems - Bridging the Gaps between Developers and Users," *Computer*, vol. 24, pp. 59-69, Apr 1991.
- [10] R. Daft and R. H. Lengel, "Organizational Information Requirements, Media Richness and Structural Design," *Management Science*, vol. 33, pp. 554-569, 1986.
- [11] J. Bragge, H. Merisalo-Rantanen, and P. Hallikainen, "Gathering innovative end-user feedback for continuous development of information systems: a repeatable and transferable e-collaboration process," *Professional Communication, IEEE Transactions on*, vol. 48, pp. 55-67, 2005.
- [12] P. Haumer, K. Pohl, and K. Weidenhaupt, "Requirements elicitation and validation with real world scenes," *IEEE Transactions on Software Engineering*, vol. 24, pp. 1036-1054, Dec 1998.
- [13] K. Holtzblatt and H. Beyer, "Making Customer-Centered Design Work for Teams," *Communications of the ACM*, vol. 36, pp. 93-103, Oct 1993.
- [14] J. Mylopoulos, L. Chung, and E. Yu, "From Object-Oriented to Goal-Oriented Requirements Analysis," *Communications of the ACM*, vol. 42, pp. 31-37, January 1999.
- [15] C. Atkinson and T. Kuhne, "Model-driven development: a metamodeling foundation," *Software, IEEE*, vol. 20, pp. 36-41, 2003.
- [16] B. Selic, "The pragmatics of model-driven development," *Software, IEEE*, vol. 20, pp. 19-25, 2003.
- [17] V. Kulkarni and S. Reddy, "Separation of concerns in model-driven development," *Software, IEEE*, vol. 20, pp. 64-69, 2003.
- [18] S. Kelly and J.-P. Tolvanen, *Domain-specific modeling: enabling full code generation*: Wiley-IEEE Computer Society Press, 2008.
- [19] S. W. Ambler, "Agile model driven development is good enough," *Software, IEEE*, vol. 20, pp. 71-73, 2003.
- [20] C. Bock, "UML without Pictures," *Software, IEEE*, vol. 20, pp. 33-35, 2003.
- [21] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *Software, IEEE*, vol. 20, pp. 42-45, 2003.
- [22] D. Moreno-Garcia and J. Estublier, "Model-driven Design, Development, Execution and Management of Service-based Applications," in *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, Honolulu, HI, 2012, pp. 470-477.
- [23] Nokia. (2005, 3/9). *Staying in Touch With Presence*. Available: <http://www.nokia.com>
- [24] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems*, vol. 24, pp. 45-77, 2007.
- [25] J. E. van Aken, "Management research based on the paradigm of the design sciences: The quest for field-tested and grounded technological rules," *Journal of Management Studies*, vol. 41, pp. 219-246, 2004.
- [26] P. Järvinen, "Action Research is Similar to Design Science," *Quality & Quantity*, vol. 41, pp. 37-54, February 2007.
- [27] S. Gregor and D. Jones, "The anatomy of a design theory," *Journal of the association of information systems*, vol. 8, pp. 312-335, 2007.
- [28] T. J. Reynolds and J. Gutman, "Laddering theory, method, analysis, and interpretation," *Journal of Advertising Research*, vol. 28, pp. 11-31, 1988.
- [29] G. A. Kelly, *The Psychology of Personal Constructs*. New York: W W Norton & Company, 1955.
- [30] C. E. Gengler, D. J. Howard, and K. Zolner, "A Personal Construct Analysis of Adaptive Selling and Sales Experience," *Psychology & Marketing*, vol. 12, pp. 287-304, 1995.
- [31] C. E. Gengler and T. J. Reynolds, "Consumer understanding and advertising strategy: analysis and translation of laddering data," *Journal of Advertising Research*, vol. 35, pp. 19-33, 1995.
- [32] M. Rossi and T. Tuunanen, "A method and tool for rapid consumer application development," *International Journal of Organisational Design and Engineering*, vol. 1, pp. 109-125, 2010.