Pekka Suopellonmäki

GUI Personalization Framework driven by Personal Semantic User Profile

Master's Thesis in Information Technology

5 November 2017

University of Jyväskylä

Faculty of Information Technology

Author: Pekka Suopellonmäki

Contact information: pekka.e.suopellonmaki@student.jyu.fi

Supervisor: Oleksiy Khriyenko

Title: GUI Personalization Framework driven by Personal Semantic User Profile

Työn nimi: Sovelluskehys käyttöliittymän personointiin käyttäen semanttista käyttäjäprofiilia

Project: Master's Thesis

Study line: Web Intelligence and Service Engineering, Faculty of Information Technology (Department of Mathematical Information Technology)

Page count: 79+5

Abstract: As Internet has evolved, the world has become increasingly connected. We use wide range of different user interfaces to interact with each other and services. While some design patterns are commonly recognized, such as gear icon for settings and downward pointing arrow for downloading a file, typical graphical user interface today does not take individuals and their preferences or restrictions into account. Instead, from service's point of view, all users are generally considered as one homogenous group and they need to adapt to each different UI separately regardless of language, culture, age etc. While common iconography is in our collective mind helping transition from one service to another, developing truly personalized interfaces is time and resource consuming. This thesis aims to describe a framework and technologies involved for graphical user interfaces that, based on portable personal semantic user profile, can adapt individually. Framework uses Semantic Web Technologies to set user properties in right context allowing them to describe his/her profile in own terms, thus mitigating language and culture differences. Likewise for services: developers can describe their application properties in semantic fashion. The result is description of GUI personalization framework which works as intermediate between user and applicable service matching properties between them using ontology alignment.

Keywords: semantic personalization, semantic user profile, ontology alignment, semantic web, GUI personalization, human-computer interaction

Tiivistelmä: Internetin kehittyessä maailma verkostoituu yhä enemmän. Käytämme päivittäin monia laitteita ja erilaisia käyttöliittymiä, mutta vaikka ne monesti jakavat yleisiä käytänteitä ja kuvakkeita, eivät ne kuitenkaan mukaudu yksittäisen käyttäjän tarpeisiin. Vaikka ihmisillä on monia eri ominaisuuksia tai rajoitteita, jotka vaikeuttavat käyttöliittymän-omaksumista, palvelun tai ohjelman näkökulmasta käyttäjät mielletään silti yhtenä homogeenisenä joukkona, jonka on mukauduttava käyttöliittymään. Omaksumiskykyyn vaikuttavia tekijöitä ovat esimerkiksi kieli, ikä, koulutustausta ja kulttuuri. Mukautuvan käyttöliittymän toteutus nykytekniikalla on kuitenkin kallista ja aikaa vievää, joten useimmat ohjelmat tai palvelut eivät sellaista edes tarjoa. Suurimalle osalle yrityksistä se ei yksinkertaisesti ole realistista aika- ja resurssipulan vuoksi, vaikka se voisi tuoda kilpailuedun muihin vastaaviin palveluihin nähden. Tämä työ pyrkii esittämään sovelluskehyksen, jolla käyttäjän semanttisen käyttäjäprofiilin perusteella voidaan toteuttaa personoitu, yhtenäinen käyttökokemus eri sovellusten välillä. Semanttisilla kielillä kuvailtu käyttäjäprofiili on helposti kuvailtava ja mukautuva, sillä käyttäjä voi kuvailla profiilinsa omin sanoin. Profiilin ja sovelluksen semanttista kuvausta vertaamalla voidaan käyttöliittymä mukauttaa soveltuvin osin henkilökohtaisesti sopivammaksi. Työn lopputuloksena on kuvaus sovelluskehyksestä.

Avainsanat: käyttöliittymät, mukautuvat käyttöliittymät, semanttiset web-teknologiat

Figures

Tables

Contents

1 Introduction

As the Internet has evolved to everyday medium for most of us, we interact with multitude of different interfaces and probably not even realizing how different they are. We look for information, consume content in form of audio or video, do online shopping and keep in contact with our friends among other things. Internet today has come far from the early days of static documents hyperlinked to each other. In past, there have been multitude of different user interface systems and design paradigms to help making human-computer interactions more and more user friendly. Person who is new to computers will probably find modern smart phone less scary than a computer with only terminal access. However, most interface systems often disregard individual user traits, and stick to standardized or de facto standardized practices, such as commonly known iconography and terms.

Graphical User Interfaces (GUI) vary depending on what content we consume or what services we use. For example, video streaming service looks and functions differently compared to service which let you book time for dentist. This is obviously by design, since the goal is very different. However, despite using common design patterns and visual language, it can be confusing for users to jump from one service to another. We are all different and computer systems do not take into consideration our personal traits that affect how we internalize information. We have varying cognitive skills, education, personality and even motivation or mood that play role on how we understand functionality of an application, thus making human-computer interaction difficult (Benyon 1993). Even two persons with same cultural and educational background may have preferences developers may not realize. Using aforementioned terminal - smart phone comparison, user may still prefer text-based terminal for some tasks even if “easier” GUI is available as typing can be faster and more productive for experienced person. Personal preferences are not only limited to application usage but information representation as well: some people prefer data to be shown in textual form whereas others understand same topic better in a graph or images. User might also have some restrictive ability, such as color blindness or wrong language that prevents him/her from using application altogether. Instead, we often must be satisfied with what designers and developers thought was best, looked nicest or was most costefficient way of developing a service as development is time and resource consuming effort. Applications and services are generally designed as one size fit for all.

Adaptive user interface as a concept has been around since early 1980s. Also, there is concept of content aware applications where representation of media is changed depending on its format. Textual data is in text format, RSS feed can be automatically stylized and CSV data is displayed as tables. However, this still does not take into consideration personal preference how one person wants to consume that information. Moreover, adaptive user interfaces are usually confined to one service. Since there is no standard on how interfaces are being built, getting user information or preferences for personalization can be hard. User must either register to service or give permission to his/her data from third party source such as Google or Facebook. People can be rightfully cautious in registering to a service they do not often use, especially if that services starts asking for seemingly invasive personal questions. Third parties, however, probably do not either have the information needed to give user best possible experience, or schema of their user data, which can change over time, does not fit to given service and thereby making UI adaptation impossible to maintain. Thus, while adaptive interface is achievable within a closed system, truly ubiquitous UI personalization across different platforms and services is hard due to unreliable user identification and limited or outdated access to relevant user information.

There have been efforts for personalizing and adapting user interfaces for individual needs (adaptive UIs), but not so much for shortening user experience gap between different applications. That is, transferring user preferences for adapting user interfaces from one service to another, thereby blurring the distinction of separate services. Semantic Web technologies could be used as common "language" between application from different vendors. According to Web Consortium standard, Semantic Web is a "web of data" (W3C 2015). It is not separate from normal web that we generally refer as Internet, but extension to it. As its name implies, Semantic Web was designed to give meaning to data. That is, to help machines to understand and reason it. Mainly this is done with Resource Description Framework (RDF) metadata model which attempts to give context and relationship between different information entities. With a proper tool, user could build his/her own semantic profile and ontology, which then could be matched against service's semantic terminology for personalized UI adaptation. Optimal solution from user's standpoint, however, would be if this profile is automatically built from data on how (s)he uses applications making framework mostly invisible and as distracting as possible. Although implementation needs to be built for each application and service, common methods and ontology alignment process gives much more equal chance for all software providers regardless of their size.

Semantic Web Technologies have been proposed for GUIs in past (Khriyenko 2015, Kadlec and Jelínek 2007, Gajos, Weld and Jacob 2010). With the rise of Internet of Things, computer-computer interaction is also becoming more important as different devices from smart cars to smart home appliances interact autonomously with each other. Semantic interfaces allow machines to talk to each other in schema-free fashion even if they are from completely different companies and work on different domains. Personal Semantic User Profile (PSUP), described in this thesis, would be based on RDF documents for handling request and provide user profile to third party applications. Machine learning and cognitive computing with systems like IBM Watson, Google Deep Mind and Microsoft Cognitive Services are used for many systems from image recognition to self-driving cars and IoT devices, but those could also be utilized for UI personalization. Machine learning regarding PSUP Framework is mentioned in chapters 2 and 3.8.

This thesis aims to give one proposal for individually personalized ubiquitous and unopinionated UI system that is easy to for service providers to implement. Thesis is constructive, although it is not based on real world working prototype. Therefore, while being descriptive in nature, everything described is based on existing technologies and protocols, thus it could be built on in later work. Thesis is structured as follows. In chapter two background and problems with current graphical user interfaces (GUI) are elaborated. Also, usable Semantic Technologies are detailed. In chapter three the structure, functionality and implementation of PSUP and GUI Personalization Framework as well as possible issues and future research ideas are provided. Thesis is concluded in chapter four.

2 Background

This chapter describes limitations of current graphical user interfaces, what problems they face with different user preferences and restrictions, UI adaptation, and what current technologies could be used to share similar user experience across web applications. That is, what technologies Personal Semantic User Profile is based on.

2.1 Confined User Experience

Today websites and web applications come in many forms. Graphical User Interfaces can vary greatly depending on what platform application runs or what it provides to the user. Even same application can look and feel totally different on other platforms due to platform limitations or design paradigms or input controls. Some devices use keyboard and mouse, others touch input or even voice commands.

A good GUI is consistent, intuitively logical, gives user the information when needed, avoids situations that can lead to an error, and provides shortcuts for more experienced users. These are some of the guidelines for good GUI design by Nielsen and Molich (Molich and Nielsen 1990, Nielsen 1994). Despite the age, especially considering how fast Information Technology and user interfaces evolve, these guidelines are still valid and should be considered when designing and building user interfaces as they will help making interface usable.

Along with usability, visual fidelity is also big part of applications’ appeal as well. Especially for making first impression when user is possibly browsing through different competing applications. As interfaces for human interaction are filled with various data, such as text, numbers, images, dates, graphs and so on, clear visual representation is crucial for service’s or application’s success. People value finished looking product. Even more so if they must pay for it. Visual fidelity does not only attract users to try, but also impacts the usability of that application. When icons and actions are set up so that user can understand resulting events without instructions, GUI is considered intuitive. It could be said that data representation plays crucial role in successful GUI.

Considering visual attractiveness and people valuing good data representation and user interfaces, it is no wonder that, to stand out from competition and thus possible market share gain, companies look for new technologies and put lots of effort to visual design. That effort, however, is usually put on catering biggest target audience or average "preference" of users. It is reasonable as designing, building and maintaining software is time and money consuming. Focusing on biggest dominant user group attributes makes sense from business standpoint. That usually being, user is able bodied, using standard input devices (Gajos, Weld and Jacob 2010), familiar with common icons, in neutral state of mind etc.

As philosopher David Hume put the old wisdom, "Beauty in things exists merely in the mind which contemplates them", visual attractiveness certainly comes down to personal preference and information can be represented in many different forms. There are even websites that solely focus on featuring different infographics. So, for example, while one finds minimal, focused infographic style appealing representation of data, others might prefer it displaying more at once. Someone else wants it in just plain text or table form without any graphical elements. Often, as developers, we find these contradicting and therefore impossible to offer at the same time. However, when it comes to GUIs users rarely have any control over data representation, but instead service provider or application developer makes the decision for them. Moreover, personal preference is not the only variable user may have that affect the way (s)he understands computer systems. Cultural background, motivation, personality, education and cognitive skills all affect the way we interpret icons and commands. Mood, motivation and goals can change many times during a single day. (Lavie and Meyer 2010).

It can be said that user interfaces are generally designed as "one size fit for all" where most of us need to adapt to use different interfaces instead of interfaces adapting to our preferences. Generally, we are used to switch workflow from one application to another even if we do not realize it.

2.2 Adaptive User Interfaces (AUI)

With varying personal preferences in mind, user interface could be adaptive. In Langley's paper User Modeling in Adaptive Interfaces AUI is defined as "a software artefact that improves its ability to interact with a user by constructing a user model based on partial experience with that user" and that improvement should be based on generalization of past experiences which in turn affect future interactions (Langley 1999). Therefore, simple memorization of user's actions is not enough for calling an UI adaptive, but instead it needs to be based on a model of a user from multiple past interactions. By this definition, adaptive system seems quite similar to the definition of machine learning, but as Langley puts it: "adaptive interface does not exist in isolation, but rather is designed to interact with a human user" emphasizing the need for constant user input. In short, AUI is a system that adapts display information and available actions depending on user's status, goals and system state.

For the record, while it could be considered as adaptation, so called responsive web layout is not considered as adaptable UI in this thesis. Even if layout would adjust to different screen sizes, it is not based on user's actions nor identity, thus user model.

In their paper Automatically Generating Personalized User Interfaces with Supple, Gajosa, Weld and Wobbrock list the following assumptions of typical computer system UI: "1) that they [User interfaces] are going to be used by an able-bodied individual, 2) who is using a typical set of input and output devices, 3) who has typical perceptual, cognitive, and motor abilities, and 4) who is sitting in a stable, warm environment" (Gajos, Weld and Jacob 2010). Simple possibility to adjust service's color scheme may seem like trivial feature for most people, but it can be great help for someone with not typical perceptual abilities such as color blindness. Also, that is not the only constraint user may have. Adjusting to every possible scenario, however, from business standpoint is not feasible as building user interface takes time as it already is. Even with able-bodied individuals, mood or current physical situation may alter the way user would want to use the service. Gajosa, Weld and Wobbrock state: "[...] with a multi-touch screen, low vision, or riding on a jostling bus may drastically hamper the person's effectiveness—not because of any inherent barrier to interaction, but because of a mismatch between their effective abilities and the assumptions underlying the interface design." (Gajos, Weld and Jacob 2010)

While AUI as a concept has been around since early 1980s, proper AUI is not very common in actual computer systems even today. Implementing adaptive system has significant cost associated with it compared to "static" interface designed for everyone (Benyon and Murray, Applying user modeling to human-computer interaction design 1993), however, well built, adapting and personalized interface can be great asset against competition even if initial cost is greater. Adaptive and adaptable user interface can increase user performance and satisfaction as visual attractiveness changes from person to person. Context awareness and personal UI can make system more approachable and intuitive for more users. (Khriyenko 2015). From pure business standpoint, adaptive user interface can be still hard to justify.

Benefits of AUI depends on many factors and is not always distributed evenly for every user and use case. Lavie and Meyer examined positive and negative effects of Adaptive User Interfaces by having participants perform driving tasks with telematic system in controlled simulation environment. Telematic system had four levels of adaptability from manual to fully adaptive (Lavie and Meyer 2010). Although their study is limited to simplified laboratory driving situation, their study suggest that intermediate UI adaption should be considered over all-or-none system. The problem with fully adaptive systems, that is, it makes more automated decision for the user, is that if user wants to perform a non-routine task, they must override adaptation system and for full AUI, it may be harder. As they put it: "Intermediate levels of adaptivity keep users involved in the task and help them become more proficient when performing both routine and non-routine tasks" (Lavie and Meyer 2010). In conclusion of their paper, the benefits of AUI differed on participants age, the type of task and how familiar the given task was (routine). Therefore, AUI does not inevitably mean better system and one should be considerate when implementing one, especially more automated system. User's cognitive workload increases significantly if faced with unfamiliar situation where system has not adapted or user needs to override systems' adaptation to perform a non-routine task.

2.3 Model-Based Interface Design

As stated in chapter 2.1 UI systems are generally designed as “one size fit for all”. One reason for it is the complexity and heterogeneity of different situations systems face. In “Introduction to Model-Based User Interfaces” by World Wide Web Consortium, sources of heterogeneity include

1. Heterogeneity of users. Users differ with respect to their preferences, capabilities, culture (e.g., speaking different languages) and level of experience.
2. Heterogeneity of computing platforms. Diversity of different device types (e.g. smartphones, desktop pc, tablets) and input capabilities (e.g. keyboard, mouse, touch, motion) with different interaction modalities (e.g., graphics, speech, haptics)
3. Heterogeneity of development tools (e.g. programming languages, toolkits, widgets, libraries)
4. Heterogeneity of working environments. (e.g. noisy environment, physical distractions in moving vehicle or other people, lack of mobile signal)
5. Variability of the context of use.

(W3C 2014)

The idea of Model-Based Design is to identify set of higher level abstraction models that are not dependent on some technology or programming language. This would allow designers and developers to focus more on general higher lever concepts and functionality of an UI from more semantic oriented level; developers should not fixate on implementation details and tools too much at first, but instead only afterwards choose tools and languages that fit in the higher conceptual level (W3C 2014).

In Model-Based User Interface Design, UI models are represented on four levels of abstraction: The Task and Domain Model, the Abstract UI, the Concrete UI and the Final UI. Starting from top, the Task and Domain Model expresses user’s goals in given domain when interacting with the UI. Abstract UI tells how the UI works in terms of Abstract Interaction Units and relationships between them, and Concrete UI is technologically independent representation of the UI. That is, for example, graphical representation of how the UI will look. When the UI model is transformed to lower levels of abstraction ("Concretization"), it reaches the Final UI level. This is the stage where UI is represented in actual language and platform specific implementation (Java, HTML etc.).

Considering the UI in multiple different abstraction levels gives benefits like

- higher level of abstraction
- consistency from early stages of development process
- step-wise development cycle with the separation of concerns can produce good base for well-structured system
- involvement of domain-experts and non-programming professionals

(W3C 2014)

There have been interface development tools that use Model-Based system. One of these were MOBI-D presented by Angel R. Puerta and David Maulsby in 1997. In their paper Management of Interface Design Knowledge with MOBI-D, it was described as "an interface-development environment that allows developers to define, organize, visualize, and edit interface design knowledge." MOBI-D environment aimed to combine different Model-Based abstraction levels in one suite of tools. (Puerta and Maulsby 1997). In essence, MOBI-D is design tool for designers and developers. Later Puerta and Jacob Eisenstein proposed adaptation algorithm for MOBI-D (Eisenstein and Puerta 2000).

While Model-Based Interface Design is not concretely applied to Personal Semantic User Profile described later in this thesis, the heterogeneity issue is still prevalent. PSUP GUI Personalization Framework proposed in chapter 3 aims to tackle issues 1 and 2.

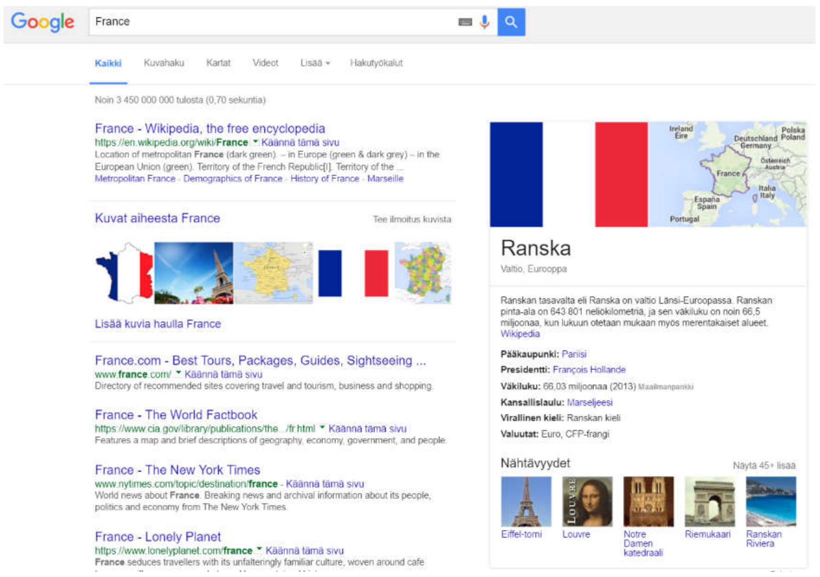
2.4 Interoperable User Interfaces with Semantic Web Technologies

This chapter describes previous work on the field and technologies PSUP GUI Personalization Framework is based on.

2.4.1 Semantic Web technologies and graphs

Word semantic means "meaning". Simply put, in information technology, instead of having simple blob of data, for example in textual form, semantic data has underlying meaning and context (Hebler, et al. 2009). That text of information is related to some other concept and this relation (context) is defined with some semantic notation. Semantic Web Technologies is linked to a concept that The World Wide Web Consortium (W3C) refers as "Web of data". According to W3C "The ultimate goal of the Web of data is to enable computers to do more useful work and to develop systems that can support trusted interactions over the network" (W3C 2015).

W3C's vision for Semantic Web is network of semantically linked data. That is, data and data stores built with extendable vocabularies and rules computers can "understand" and reason. Reasoning means that, depending on how data is linked, computer can autonomously draw conclusions and generate new information from a set of linked data. This is useful for integrating context related data to single view. For example, Google uses its' Knowledge Graph to enrich search results by trying to answer user's question with context relevant information directly in results view (see Figure 1).Moreover, when Google's search algorithm reasons user's search query, it attempts to directly answer in a way that feels more human-like interaction instead of keyword based list what we normally expect from search functionalities on Internet. Recent study by Google and Emory University showed that answer-like result is appreciated, but accuracy of that answer is also direct indicator of user satisfaction. "Focusing on answerlike results, in a controlled lab study, we identified that increased scrolling past answer and increased time below answer are signals of user dissatisfaction with answer results" the study concludes (Lagun, et al. 2014). So, while semantic data is already in use of some high-profile services, and is proven to be technically working, user adoption may not be as high as one could expect. This could be partially on how we are collectively conditioned to use computers: we think we give exact commands and expect specific answers, but sometimes exact meaning of user's intent is hard to determine. Study could also indicate that Google's Knowledge Graph is simply not yet all-encompassing enough to give meaningful answers. If answer-like result is not relevant or accurate enough, user would rather skim search results themselves. and then answer-like result might be considered hindering. Still, especially with help of machine learning methods, human-computer interaction is becoming more and more context specific and semantic accuracy is important.



Noin 3 450 000 000 tulosta (0,70 sekuntia)

France - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**France** Käännä tämä sivu
Location of metropolitan **France** (dark green) – in Europe (green & dark grey) – in the European Union (green). Territory of the French Republic[I]. Territory of the ...
Metropolitan France - Demographics of France - History of France - Marseille

Kuvat aiheesta France Tee ilmoitus kuvista

Lisää kuvia haulla France

France.com - Best Tours, Packages, Guides, Sightseeing ...
www.**france**.com/ Käännä tämä sivu
Directory of recommended sites covering travel and tourism, business and shopping.

France - The World Factbook
https://www.cia.gov/library/publications/the.../fr.html Käännä tämä sivu
Features a map and brief descriptions of geography, economy, government, and people.

France - The New York Times
www.nytimes.com/topic/destination/**france** - Käännä tämä sivu
World news about **France**. Breaking news and archival information about its people, politics and economy from The New York Times.

France - Lonely Planet
https://www.lonelyplanet.com/**france** Käännä tämä sivu
France seduces travellers with its unfalteringly familiar culture, woven around cafe

Ranska
Valtio, Eurooppa

Ranskan tasavalta eli Ranska on valtio Länsi-Euroopassa. Ranskan pinta-ala on 643 801 neliökilometriä, ja sen väkiluku on noin 66,5 miljoonaa, kun lukuun otetaan mukaan myös merentakaiset alueet.
Wikipedia

Pääkaupunki: Pariisi
Presidentti: François Hollande
Väkiluku: 66,03 miljoonaa (2013) Maailmanpankki
Kansallislaulu: Marseljeesi
Virallinen kieli: Ranskan kieli
Valuutat: Euro, CFP-frangi

Nähtävyydet Näytä 45+ lisää

Figure 1. Google uses Knowledge Graph to enrich search results by providing context related information for user.

In Semantic Web, context for data is represented in triplets: subject, predicate and object. Predicate tells how a subject is related to an object. Object can be literal value, so called data property, or another object. For example, subject which is of type "car" has predicate "brand" and value of that predicate might be "Ford". In this case, value would be literal string. Data properties can be basic data types such as integer, float, date etc. and do not have any sub properties. However, Ford could also be predefined object with predicates of its' own, such as "country of origin" or "number of employees" etc. Thus, previously mentioned subject car could have object property "manufactured by" which links Ford to it as an object instead of a data property. Eventually these different data nodes build up to a network, a graph of data where every data node has linked to other entities. Said example is described in Figure 2. Literal values are in boxes, objects in circles. Data and object relationships (predicates) are shown with arrows.

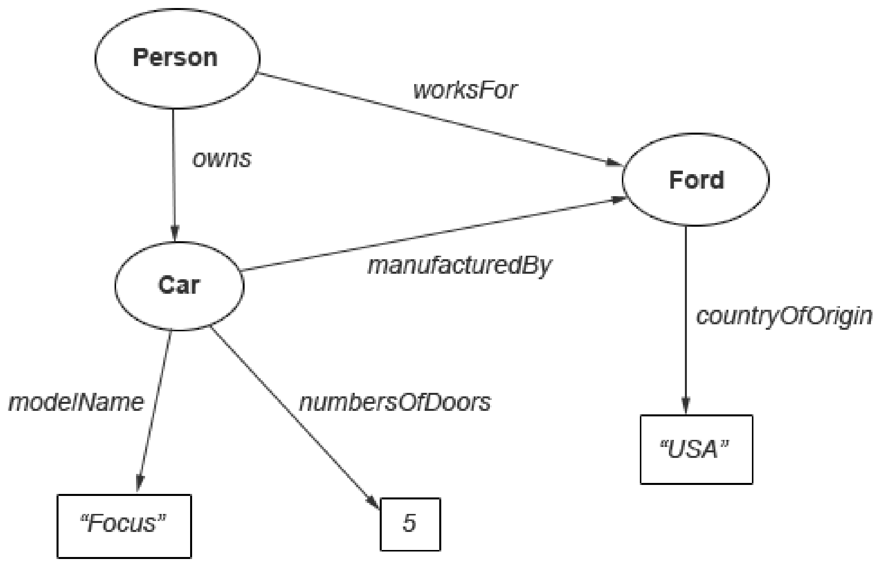


Figure 2. Example graph of Semantic web

Vocabulary of a graph is defined in semantic ontologies. Terms for predicates and classes are up to a developer. There are of course some recommended naming conventions, such as one by Open Semantic Framework (Structured Dynamics LLC 2014), but just like programming best practices in general, they can be dictated by company or personal preferences. Developer should just be internally consistent when deciding terms in ontology. Ontologies are explained in further detail in chapter 2.4.2 Ontologies.

With Semantic Web, every piece of information is linked via relation (defined in ontology vocabulary) to some another entity with unique URI. One way to understand semantic objects and data is to think single website as semantic object and its contents as separate literal values separated by markup. If a hyperlink points to a location on same document, its type can be thought as literal, whereas hyperlink pointing to other sites can be thought having object value. Outbound link ("object") can have its own literal values (content) and object values (hyperlinks to other sites). Main difference between Semantic Web and traditional hyperlinked web-documents is that hyperlink does not understand the relation between interlinked documents. It just points to another URL and computer cannot know type of that connection.

For Artificial Intelligence, nothing exists if it cannot be represented (Guarino, Oberle and Steffen, What Is an Ontology? 2009), thus, by representing data in knowledge models, such as graphs, machine can be explained how to "understand" concepts and relationships. In the example above, car's "model" is a something that relates to data entity "a car", or for another example, predicate "first name" relates to subject "person". Since all data is connected in Semantic Web, new data can be generated from existing via automatic reasoning using artificial intelligence. For example, to get the first name of an acting president of the country where one particular car model was manufactured is multistep process ranging across different domains. It is likely that information of presidents are not found in the same dataset as car manufacturers, because the domains are very different. Human, assuming having no prior knowledge, would need to search answer for each step one at a time: what company has manufactured this car, where that company is located, when this particular car model was made or when this model was manufactured, who was the president of that company's country at the time, what is his/her first name. Semantic reasoning happens the same way. Since all data is related to another, machine can do all these steps automatically. This could be even done as a single SPARQL query. However, just like spoken languages, same word can have different meaning in different context, and same concept can be defined with different terms. Different computer systems may use different word for same or similar concepts. For systems to be able to co-operate, they have to share same understanding of things. In Semantic Web this is done via Ontology Alignment, which is explained in chapter 2.4.5.

Linked data is built with technologies such as RDF, SPARQL, OWL, and SKOS (W3C 2015). In this thesis, SKOS (Simple Knowledge Organization Systems) is not used, but in short it is attempt by W3 Consortium to build standards and specifications for Knowledge Organization Systems (W3C 2012).

2.4.2 Ontologies and RDF

Term “ontology” has different meanings depending on what context or field it is used (Guarino and Giaretta, Ontologies and knowledge bases towards a terminological clarification. 1995). In general, when used with capital “o”, it refers to philosophical discipline that studies nature and structure of reality independent of any other considerations (Guarino, Oberle and Steffen, What Is an Ontology? 2009). As Guiarino et al puts it: “For example, it makes perfect sense to study the Ontology of unicorns and other fictitious entities: although they do not have actual existence, their nature and structure can be described in terms of general categories and relations.” Ontology as a philosophical discipline is closely related to Epistemology, which is a branch of philosophy that studies sources and nature of knowledge (Guarino and Giaretta, Ontologies and knowledge bases towards a terminological clarification. 1995).

In computer science, an ontology refers to a formally structured information object or computational artifact that holds relevant entities and relationships for a domain (Guarino, Oberle and Steffen, What Is an Ontology? 2009). So, in context of Semantic Web Technologies an ontology can described as a cross linked “vocabulary” for some application, domain or context in general. This “vocabulary” describes the elements for computer systems to reason with, as for AI systems things that cannot be represented do not exist (Guarino, Oberle and Steffen, What Is an Ontology? 2009). As described in chapter 2.4.1, concepts in Semantic Web Technologies are described in statements of triplets: subject, predicate, object. Predicates indicate how an object relates to a subject.

There are multiple different markup languages for defining ontologies. For this thesis, however, only two have major importance: RDF and OWL. RDF stands for Resource Description Framework and is technically not a markup language, but instead standard model for data interchange (W3C 2004). RDF is used to represent information and data relationships in the Web. More specifically any information with unique uniform resource identifier (URI). In semantic statement, i.e. subject – predicate – object, any part is a resource that can be represented with URI. Object, however, can also be literal. That is, some literal value in form of number, text, date etc.

RDF Schema is W3 Consortium recommendation. It defines basic data modelling vocabulary for describing RDF data, including ontologies. Concepts such as classes and properties are described in this recommendation, and practically every other ontology is based on this schema. XSD stands for XML Schema Definition and is similar to RDF except, as the name states, is recommendation for describing structure and constraints for XML-documents. However, it is used in RDF-documents to describe datatypes such as strings and integers. W3 Consortium also has recommendation for actual ontology language. Derived from the DAML+OIL Web Ontology Language, OWL (Web Ontology Language) is extension of RDF with much more properties for describing object relations. OWL is not one language but instead three separate sublanguages. In order of increasing expressiveness, these languages are OWL Lite, OWL DL and OWL Full. Increasing expressiveness means that every ontology in OWL Lite is also valid in OWL DL, and every ontology in OWL DL is also valid in OWL Full, but not other way around. Ontology in OWL Full may contain properties that are not defined in OWL DL. Because of lacking definitions, only OWL Full is automatically fully compatible with RDF. Therefore, OWL as RDF extension is limited to OWL Full. W3C OWL Recommendation states "OWL Full can be viewed as an extension of RDF, while OWL Lite and OWL DL can be viewed as extensions of a restricted view of RDF. Every OWL (Lite, DL, Full) document is an RDF document, and every RDF document is an OWL Full document, but only some RDF documents will be a legal OWL Lite or OWL DL document" (W3C 2004). This can cause confusion and requires care when implementing or switching from one OWL sublanguage to another.

Simple ontology consisting of three classes (Person, Product and Car), three custom predicates (hasFirstName, ownsProduct and hasDoorCount), as well as their relations are visualized in graph form using basic RDF- and XML-Schema-classes in Figure 3. There are few simple observations from this graph that illustrates the basic idea of semantic triplets. A triplet where subject is an instance of Person class can have predicate hasFirstName, because rdfs:domain of that predicate is set be Person,. Object of that triplet would be string literal (e.g. "John"), because rdfs:range for preidcate hasFirstName is xsd:string. Also, Person can own instance of Car class even though rdfs:range for ownsProduct predicate is Product. This is because Car is subclass of Product and therefore inherits all properties in Product class. However, hasDoorCount is not a property of Product, but property of Car only.

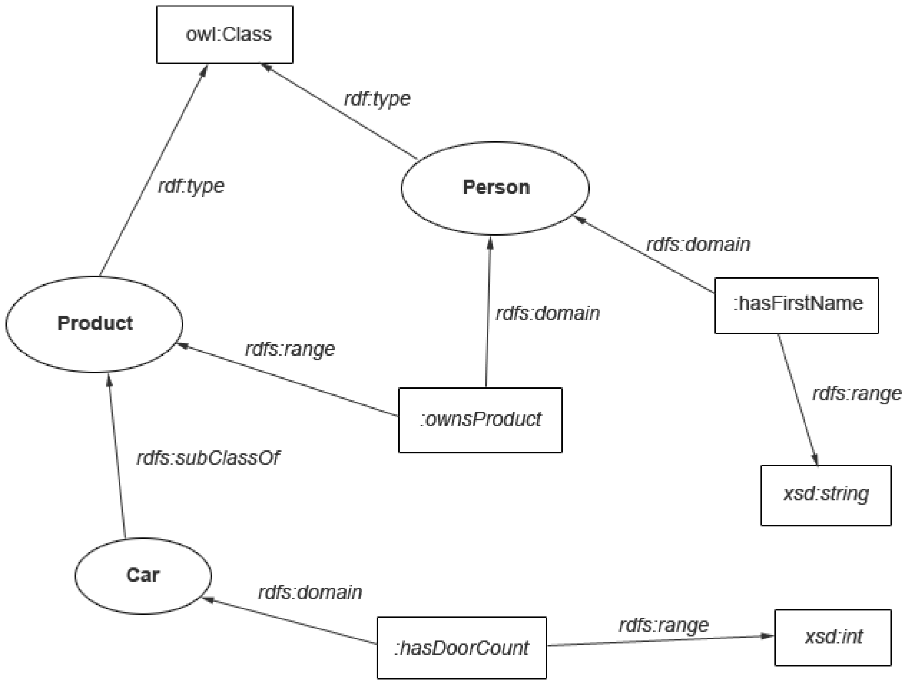


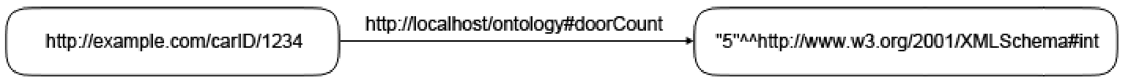
Figure 3. Example of graph described in RDF Schema vocabulary

Graphs can be described in multiple different ways and notations. Few of these are presented in Figure 4. First, simple triplet-statement is written in English, then divided in semantic triplets, visualized as RDF graph and finally in RDF format (Turtle notation and RDF/XML). Note that all nodes and edges are described as URIs but Turtle notation allows defining prefixes for better readabilty. For RDF/XML some syntax highlighting is provided

1. "The car has 5 doors."

2. Subject: http://example.com/car/1234
   Predicate: http://localhost/ontology#doorCount
   Object: "5"^^http://www.w3.org/2001/XMLSchema#int

3.


http://example.com/carID/1234
http://localhost/ontology#doorCount
"5"^^http://www.w3.org/2001/XMLSchema#int

4.
```
@prefix : <http://localhost/ontology#> .
@prefix cars: <http://example.com/carID/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

cars:1234 :doorCount "5"^^xsd:int .
```

5.
```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ns0="http://localhost/ontology#">
  <rdf:Description rdf:about="http://example.com/carID/1234">
    <ns0:doorCount rdf:datatype="http://www.w3.org/2001/XMLSchema#int">5</ns0:doorCount>
  </rdf:Description>
</rdf:RDF>
```

Figure 4. Statement described in plain English (1), RDF triplets (2), RDF graph (3), RDF Turtle notation (4), RDF/XML notation (5).

2.4.3 Open vocabularies

Everyone can create their own vocabularies for their domain specific scenarios, however, some entities are very general, and this can lead to redundant duplicates and ontology mismatches. For example, a developer describes Person class in his/her ontology, but then uses semantic data in which where Person objects are represented according to another ontology, and now there may be a need to describe same features of a person in two different terms. That is, one ontology has defined predicate "hasFirstname", while the other has virtually the same property as "firstname". One way of reducing the possibility for this to happen, is to use one of many open linked vocabularies such as FOAF. FOAF describes vocabulary for people, groups, organizations and documents. It also has addition for social web connections. Other linked vocabularies include the Dublin Core Metadata Element (DCE) for resource description (metadata), Machine Learning Schema (MLS), Simple Knowledge Organization System (SKOS) for sharing and linking knowledge organization systems and Creative Commons Rights Expression Language (CC) for describing copyright licenses in RDF.

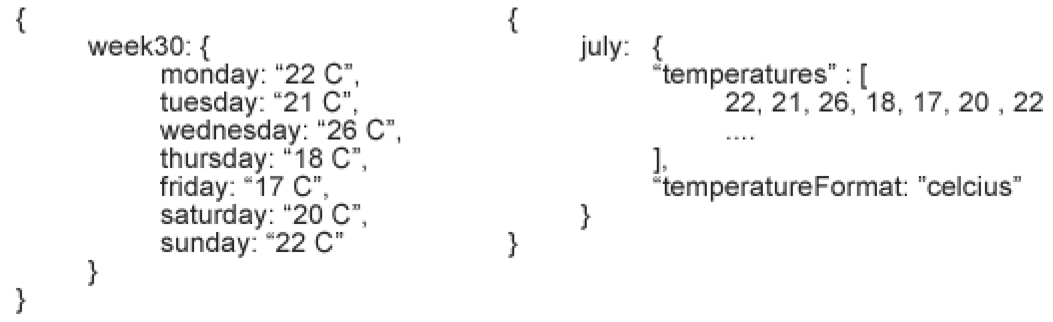
However, when using different ontologies, ontology mismatches are often unavoidable. Therefore, Ontology Alignment methods are used to link set of properties from different ontologies with varying degree of similarity. Ontology Alignment is described in chapter 2.4.5.

2.4.4 Data, knowledge and queries from semantic data

Semantic Web works under Open World Assumption (OWS). Open World Assumption states that if some statement is not clearly defined or deductible (reasoned) from known data, we cannot say if it is true or not. It can be only said that it is unknown and we can't assume either state. In Close World Assumption (CWA), an unknown state would default to false.

Both assumptions, open and closed world, have their benefits. Closed World Assumption reduces the complexity of problems by narrowing the options developer can make. Program working under CWA assumes the world is static and thereby easier to model. If state of something is unknown, developer does not have to worry about it as much as in OWS. If new options arise, existing programs don't react to the them at all, but instead work as they worked previously. This means, however, that if a program is depending on other interfaces, changes to these interfaces can cast the program dysfunctional. For example, a weather application reading feed from third party weather API will stop working properly if the API does not work or is completely removed. In CWA, developer can prepare in situation like this by programming a fallback interface to another API should the first one to not respond. This may seem like a smart, adapting program, but is still very much static approach to changing environment. Application itself does not adapt, because it has only predefined solutions to possible problems. Said weather application will still render useless if second API is also not responding or if the interface changes on first API.

As stated, Closed World Assumption is static, but easier for developers to comprehend because it reduces complexity. Open World Assumption, however, is more true to the real world: even if something is false or unknown now, it can be true later. Because of this, applications working in OWA are inherently more complex. Applications in OWA should be able to adapt in changing environment. Using the weather application as example, API providing weather data can change over time, for example, by changing the data model or structure (see Figure 5). Application capable of working under Open World Assumption is still able to find required information from changed data structure. And even if the required data is no longer available from the API, application is capable of finding alternative sources on its own, without predetermined alternative API interfaces. From developers point of view, this is immensely more complex situation if problem is approached in traditional way.



```
{
	week30: {
		monday: "22 C",
		tuesday: "21 C",
		wednesday: "26 C",
		thursday: "18 C",
		friday: "17 C",
		saturday: "20 C",
		sunday: "22 C"
	}
}
```

```
{
	july:	{
		"temperatures" : [
			22, 21, 26, 18, 17, 20 , 22
			....
		],
		"temperatureFormat: "celcius"
	}
}
```

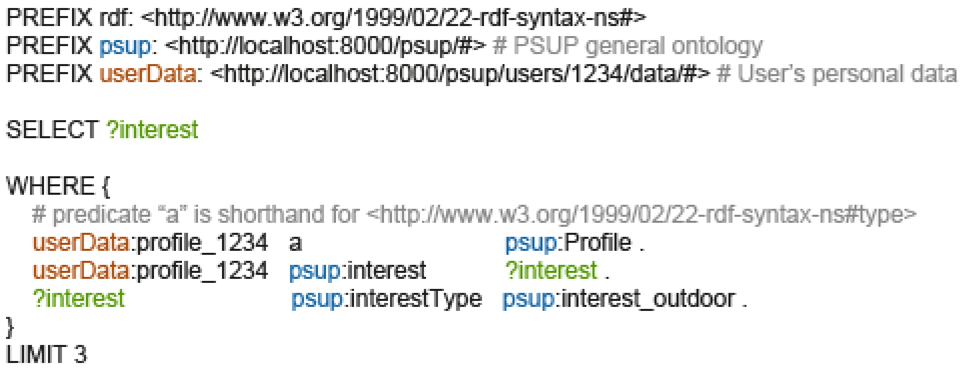
Figure 5. Similar data can be formatted in different structures.

There is distinction between data and knowledge. Traditionally, in computer systems, data has been readable for both humans and computers, but understandable for humans only. Computer program must be specifically told how to read information displayed in Figure 5. If the program is instructed to display property “monday" inside object “week30”, but is then provided with information seen on right, it will cause an error. Even if the computer is told to display “first element of first object”, it will get an array instead of string as on the left. Semantic Web Technologies, as described in previous chapters, allow computers to reason information due to fact that it can “understand” concepts such as temperature and other keywords found in data. Even different data types can be reasoned and possibly transferred.

Semantically data can be stored in many formats (see 2.4.2). OWL with Turtle notation is good format as it is readable by machines, but also by humans. Compared to RDF/XML that has lot of additional markup, Turtle notation is simpler and faster to read by human, because it is simple subject-predicate-object sentences. For example, of Turtle notation and RDF/XML see Figure 4. Other formats include JSON-LD, RDF/JSON.

Queries from RDF data can be done with languages like SPARQL. Like RDF, SPARQL is recommendation by W3C. The recommendation describes SPARQL as follows “SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports aggregation, subqueries, negation, creating values by expressions, extensible value testing, and constraining queries by source RDF graph. The results of SPARQL queries can be result sets or RDF graphs.” (W3C 2013) There are also other query languages for graph data such as GraphQL and OpenCypher.

SPARQL is similar to popular SQL-languages such as MySQL. Generally, query consists of PREFIX declarations (not necessary), SELECT section, which declares desired result set, FROM section, WHERE section, in which the restrictions to selection is defined in triples, and lastly modifiers which can alter the result set. Modifiers are comparable to other SQL-languages. That is, clauses such as ORDER BY, DISTINCT, LIMIT etc. In Figure 6 an example of query is shown. This query looks for maximum of three outdoor interests in a PSUP profile.



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX psup: <http://localhost:8000/psup/#> # PSUP general ontology
PREFIX userData: <http://localhost:8000/psup/users/1234/data/#> # User's personal data

SELECT ?interest

WHERE {
  # predicate "a" is shorthand for <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  userData:profile_1234  a                  psup:Profile .
  userData:profile_1234  psup:interest      ?interest .
  ?interest              psup:interestType  psup:interest_outdoor .
}
LIMIT 3
```

Figure 6. SPARQL example query.

2.4.5 Ontology alignment

Concepts and context is defined in ontologies as described in previous chapters. However, since vocabularies or languages can vary, ontologies may be defining same or similar concepts in different words. Single ontology cannot represent everything. For example, one ontology has defined the person who wrote a book as "author" but in other ontology it is defined as "writer". This can cause problems or unnecessary duplicates of same data in applications that use both ontologies, or user may be left into impression that writer of a book is not found just because that data entity is named differently than in other books. Therefore, there must be a process to pair these concepts between ontologies. This process is called Ontology Alignment. Or as Fernandez et all puts it, "Ontology alignment is the process of bringing ontologies into mutual agreement by the automatic discovery of mappings between related concepts." (Fernandez, et al. 2013). In short, mappings simply declare that a concept in one ontology is identical, or in some degree comparable to another concept in second ontology.

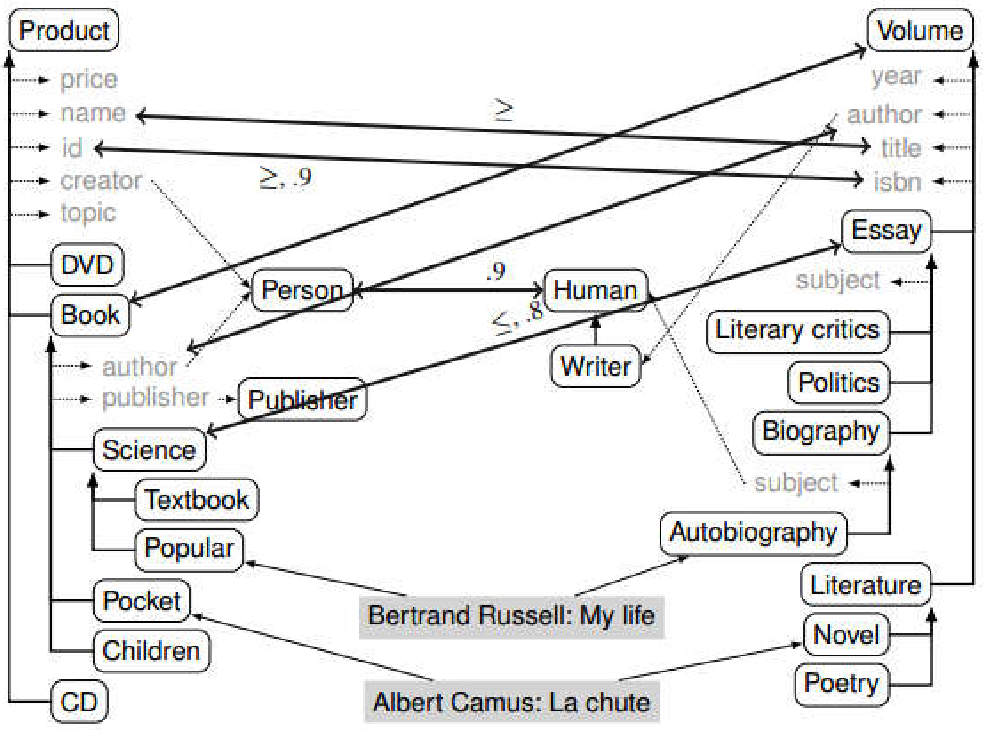


Figure 7. Two ontologies aligned. Similarity is shown as number. Image from book Ontology Matching (Euzenat and Shvaiko 2007).

Just like in real languages, multiple words can be used to define similar things, but there may be small differences in the semantic meaning. For example, designer can say to a client that some user interface is “clean” or “simple” quite interchangeably, but even though those adjectives are loosely similar, when talking about graphical user interfaces, they are not synonyms. Interface can be “simple” by showing only most relevant tools by default, whereas “clean” interface may prefer small set of colors, generous margins and in general tries to avoid too many distracting elements, be it icons, images or whatnot. Therefore, we cannot declare these adjectives to be same as one to one. Also, one could argue that, in this case, “simple” is referring to functionality of an UI and “clean” to its’ looks and therefore they also cannot be the same. To iterate, in this example, semantic meaning of these adjectives are viewed from designer’s point of view and that defines the context. Client, however, has different personal experience to words and can interpret it differently if (s)he is not familiar with terminology in certain domain. To him/her, term “clean” is used for example, to describe current state of a coffee cup or room. This leads to deduction that same word can mean something different not only in another context and domain, but also from person to person. Thus, in theory, there is always small amount of uncertainty when matching ontologies.

Because meaning of words and concepts vary in different domains, ontology matches are not strict Boolean type analogues where one concept is exactly same as other, or none whatsoever. Ontology matches are probabilities where similarity of two properties are defined from zero to one (see Figure 7). Zero being no likeness at all, and one exact same. Two properties may be exactly same or very close if they are literally same in both ontologies (e.g. “car” and “car”), they are synonyms (“universe” and “cosmos”), one is shortened version of other (“telephone” and “phone”) or they are just translations (English “car” and French “voiture”). Computer systems, need to be given certainty threshold before it can make assumption of match. Alignment process is shown in Figure 8. Matching takes in two ontologies and parameters (weights or thresholds) and possibly additional resources for determining context domain (e.g. from thesauri, WordNet). Matching result is a set of alignments where every alignment (id, e_1, e_2, r, n) consists of id, given entities (e_1, e_2), relationship between them ($r = \{=, \equiv, \leq, \geq, \perp\}$) and similarity value ($0 \leq n \leq 1$).

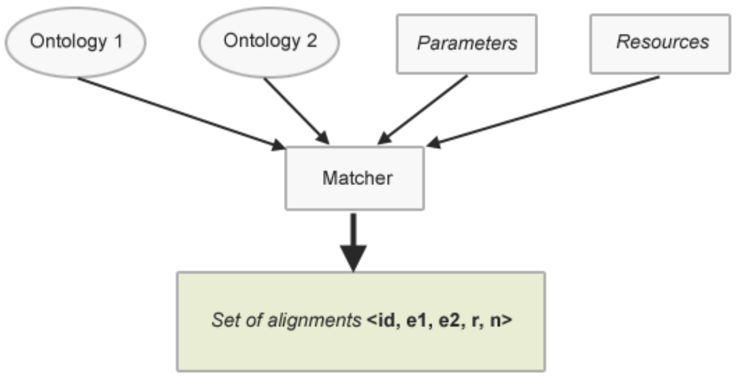


Figure 8. Ontology Alignment process.

There are multiple different ways to match classes, properties and relationships. Simplest way for most accurate alignment is to have user involvement in the process. That is, human makes the decision how similar two concepts are. This, however, requires proper tools and knowledge in domain and preferably also in semantic technologies as well. Given the vast amounts of data available, pure human made alignments are not feasible unless the domain and ontologies are very limited. Another way is to use automation with different algorithms for determining similarity or dissimilarity of ontologies. These processes can be divided into Element-level matchers and Structure-level matchers (Otero-Cerdeira, Rodríguez-Martínez and Gómez-Rodríguez 2015). In other sources these approaches are also called Schema- and Instance-based systems. Both approaches can also be combined to mixed systems (Euzenat and Shvaiko 2007). Structure-levels matchers look for similarity in ontology or data structure whereas in Element-level matchers individual entities are analyzed in isolation regardless of structure. Both, Element-level and Structure-level matchings can be combined. Different matchings can be run sequentially or in parallel for more accurate results.

Element-level matching can look for character by character similarity or dissimilarity of two words. That is, how many same characters two strings (e.g. two predicates) have. String-based techniques include Hamming distance (dissimilarity), Substring similarity, N-gram similarity, Edit distance (dissimilarity), Jaro measure (similarity), Jaro-Winkler measure (similarity) and Jaccard (similarity), which all compare strings by characters or tokens (strings divided into groups of n characters). Sting-based approaches have algorithms for document-level matching too. For example, algorithms for comparing term frequency in two documents. Note that documents are considered as isolated entities and therefore belong to element-level matching.

Beside string based methods, word similarity is measured also in linguistic and phonetic approaches. Euzenat and Shvaiko describe them in Ontology Matchin as "Extrinsic linguistic methods use external resources, such as dictionaries and lexicons. Several kinds of linguistic resources can be exploited in order to find similarities between terms." (Euzenat and Shvaiko 2007). With linguistic approaches, word relationships can be determined beyond literal similarities. For example, WordNet is lexical database of English language. It can be used to find synonyms and similar meaning of two words, but opposites as well (e.g. theory is opposite of practice). Language based methods are also used to find synonyms in other languages (multi-lingual lexicons). Soundex is algorithm for measuring similarity of two words by how they sound in English.

Structure-level (Schema-based) matching techniques mostly considers the arrangement of data disregarding element values. Internal structure refers to properties of entities such as datatypes and domains. However, properties and ranges of different ontologies can be numerous (e.g. different ontologies can use different datatypes for similar data) and therefore they must be used with other matching approaches. (Euzenat and Shvaiko 2007) That is, simple comparison of datatypes does not give enough information for semantic similarity of two entities, but it can help to evaluate likelihood of it. Consider entities "personAge" and "personName". String-based approach can give close relation (distance) as both of them include term "person". However, former datatype is integer whereas latter is string, thus, likelihood of them being same may not be likely. Matching with internal structure can be used to elimination.

Comparison on entities based on their relation is called relational structure comparison. As ontologies can be viewed as graphs where edges are labeled by predicate names, entity matchings can be done with various graph algorithms by calculating distances of entities (nodes in graph). Considering different shapes of graphs (varying depth and width), and whether they have enclosed loops or not, related classes may have similar graph structure.

Mere similar graph structure can be considered rather loose conjunction between entities and therefore taxonomic structure (i.e. graph using subClassOf relation) has been more popular among researchers (Euzenat and Shvaiko 2007). Simplest method is counting the numbers of edges between entities (classes). Relationships can be used to find similar classes by their child elements and properties. E.g. classes Book and Article can have similarity rating based on both having similar property "writer" and equivalent "author", both subproperties of higher level element "creator". Mereological structures (entities part-of relationship) can also be used computing similarity between classes, however, according to Euzenat and Shavaiko, it is not easy to find properties that carry this kind of structure .

Ontology alignment can be done via deduction and implications (reasoning). This can be done from description logic (e.g. father = man with at least one child) or with external ontologies. External ontologies work as intermediators providing context between ontologies to be aligned (Euzenat and Shvaiko 2007).

This chapter described some approaches to ontology alignment. Ontology alignment is crucial technology within Semantic Technologies as it allows finding similarity between vocabularies. PSUP Framework requires this. Furthermore, information in semantic format has inherit knowledge. Machine can reason (that is, "understand") data because the concepts in data is explained to it. One widely accepted definition for machine learning is that machine can be said learning if it improves its performance on a task from experience (Mitchell 1997). Given there is enough previous alignment data, machine learning could be utilized to automatically generate new and improving alignments from additional ontologies without human interaction. Thus, generating deeper and more precise matchings compared to algorithms run once. This possibility is discussed more in chapter 3.8 Future research.

2.4.6 SSWAP

All Semantic Technologies mentioned in previous chapters can be utilized with Simple Semantic Web Architecture and Protocol. As the name states, SSWAP is architecture and protocol for transferring heterogenous semantic data between services. It allows service providers to describe their available resources for other to discover and use. In their paper "SSWAP: A Simple Semantic Web Architecture and Protocol for semantic web services" Gessler et al. list the following hindrances with current technologies:

1. Fatal mutability of traditional interfaces: if service provider change their interfaces clients using that will fail altogether.
2. Rigidity and fragility of static subsumption hierarchies: changing a property or a class close to the root of ontology hierarchy, will alter all subsequent classes.
3. Confounding content, structure, and presentation: the data of value is not easily readable, if at all, when it is entangled with its data structure or presentation layer.

(Gessler, et al. 2009)

These limitations are related to the fact that applications and services are generally always in evolving state. Data structure is often set in early stage of development and changes likely occur during the process. With API URIs problems can be reduced by serving old URIs alongside new ones, but should the interface grow more, this will cause issues with maintainability and software rigidity. That is, due to bloated API, software or service is not able to change to meet new demands (Gessler, et al. 2009). Moreover, old API may have to be removed altogether in favor of new one due to security reasons. If old API utilizes deprecated technologies, new security issues may not be feasible to fix the for lack of resources or other business reasons. Therefore, clients using an API cannot trust for it to stay unchanged.

SSWAP aims to tackle problems of serving ontologies over Internet. Often ontologies for semantic web services are in single huge file. Terms within ontology are pointed via fragment identifier (#). For example, http://localhost/psup/user_ontology/1234/#firstname. However, these fragment identifiers are client-side pointers and not necessarily sent to servers. Therefore, separate requests for different terms in a single ontology would result server to send same complete ontology file every time. This causes unnecessary use of net bandwidth and delay (D. Gessler 2014).

Architecture-wise SSWAP system has four elements: service providers, clients, discovery servers and ontologies. SSWAP is based on RDF technology, but also in RDF's conceptual model of how information is presented in the form of entities and relationships known as RDF-triples (Gessler, et al. 2009). For description of RDF-triplets, see chapter 2.4.2. SSWAP itself is an ontology as well. It is ontology specifically created to for web resources to describe themselves, find and semantically encode them. “SSWAP defines terms such as what it means to be a web resource, who provides that resource, and how the resource maps its input to its output. SSWAP does not define the particulars of the resource — such as if 'gene' stands for REV7 or Gene Hackman or Gene Simmons — but it enables you to do so.” (D. Gessler 2014).

Data flow with SSWAP is simple and involves three RDF documents: Resource Description Graph (RDG), Resource Information Graph (RIG) and Resource Response Graph (RRG). Every resource provider has RDG document from which requests are informed what type of input and output that provider handles. Provider may only have one endpoint to which it answers to. RDG (as RIG and RRG as well) is described in OWL and therefore required input and type of output is also machine readable. From the information in RDG, a Resource Information Graph (RIG) can be generated either automatically or by human filling a form of required information described in RDG. On receiving RIG-document, provider runs the required logic and calculations (or sends its own requests to other SSWAP services) and responds with RRG. Depending on whether a request to provider has any additional information (RIG payload), provider either respond with RDG document or RRG document. The data flow is illustrated in Figure 9. All mentioned documents are basically the same, but named differently depending on how complete that document is. That is, during SSWAP service engagement, one can think of just one document being sent back and forth between client and provider.

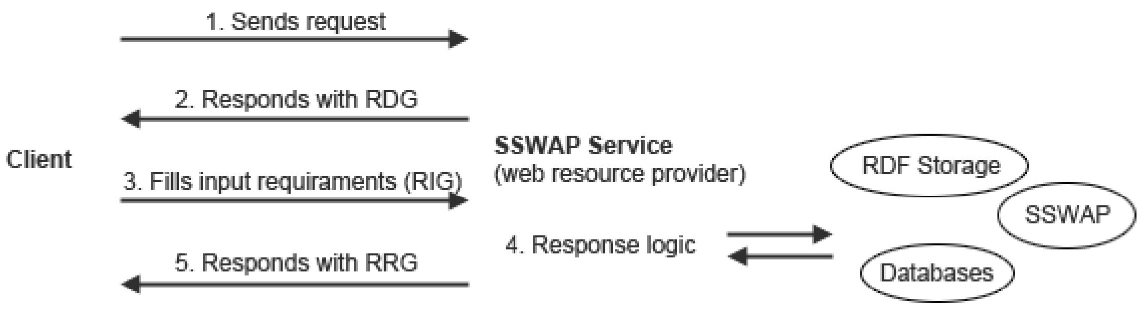


Figure 9. Data flow in SSWAP engagement

With Open World Assumption in mind, it is sensible to assume that PSUP is not the only service providing semantic user information. Therefore, PSUP framework should work as SSWAP service, thus making PSUP discoverable and giving different applications a choice whether to use PSUP profiles or not. Figure 10 shows how profile request to PSUP service could be exposed as SSWAP service.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix sswap: <http://sswapmeet.sswap.info/sswap/> .
@prefix psup: <http://localhost:8080/PSUPService/ontology/PSUPServiceOntology.owl#> .
@prefix resource: <http://localhost:8080/PSUPService/PSUPServlet> .

resource:PSUPServlet
   rdf:type sswap:Resource, psup:PSUPLoginService;
   sswap:providedBy resource:resourceProvider ;
   sswap:name "PSUP Login service" ;
   sswap:oneLineDescription "Accepts profile request and provides matching user profile." ;
   sswap:operatesOn [
      rdf:type sswap:Graph ;
      sswap:hasMapping [
         rdf:type sswap:Subject, psup:ProfileRequest ;
         psup:userId "" ;
         psup:authToken "" ;
         sswap:mapsTo [
            rdf:type sswap:Object, psup:ProfileResponse;
         ] ;
      ] ;
   ] .
```

Figure 10. Discoverable PSUP SSWAP service

2.4.7 Other technologies used in PSUP

This chapter describes few other noteworthy technologies for PSUP Framework.

“The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf." (Group 2012) If, for example, site A, where user has account to implements OAuth model, a third-party site or service B can ask user to use his/her site A account credentials to login or authorize use of his/her data in site B. For average consumers OAuth is currently most notable by the widely popular use of Facebook and/or Google logins; services often ask user to login with his/her Facebook or Google credentials instead of creating new account just for that single service. Although it can cause some privacy concerns, in general this can be useful for both, the service and user, as it is one less account user must remember password to and services do not have to store sensitive user data on their servers. If services just want to identify user but not to worry about securely storing passwords, this can be ideal. In practice, however, services usually need to also offer normal account creation for their service, because not all their user base have, in this example, Facebook or Google account, and neglecting those users is bad business practice.

User credentials are not exposed to third party applications, because only “certificates” known as access tokens are provided to the requesting application. Data flow of OAuth authentication would be as follows:

1. Application A asks user to authorize the use of his/her data on service B.
2. User is prompted to login to service B and (s)he receives access token, which is a random string and preferably expires in some time. Some services also provide refresh tokens that can be used to automatically generate new access token to replace expired one.
3. Application A receives the access token and asks for user information (to which access token gives access to) and provides the access token to service B along with the request.

4. Service B can verify that the token is valid, not expired and grants access to requested data. If valid, returns the requested information.

OAuth 2.0, created first in late 2006, is sequel to previous OAuth standard. While 2.0 is not directly backwards compatible, it holds on to similar approach and architecture to OAuth 1.0. (Aas 2013). However, it is considered as more complex and interoperable. (Hammer 2012). Nevertheless, with support from companies like Twitter and Facebook, Google and Microsoft, OAuth 2.0 has become one of the most popular standard for API authentication of today. Status of OAuth 2.0 by IETF (Internet Engineering Task Force) is “Proposed Standard” meaning it is not final but stable and well-reviewed specification. However, despite its popularity and fast adaptation, OAuth2 is not without its flaws. OAuth2 and previous WRAP (Web Resource Authorization Protocol) specification dropped client-side cryptography requirement in favor for SSL/TLS and thus more straightforward implementation. This opens the protocol for human based errors and phishing vulnerabilities. Subbu Allamaraju, author of the RESTful Web Services Cookbook explained it in a private note: “If a client application sends a request to an erroneous address (‘mail.exmple.org’ instead of ‘mail.example.org’), the rogue server at "mail.exmple.com" now has the client access token and can access its mail.” (Dubray 2010) This means that part of the protocol security is moved from protocol itself to developers implementing it and therefore relying developers are not malicious nor make mistakes. Eran Hammer, one of the early contributors for OAuth states in his leaving note: “To be clear, OAuth 2.0 at the hand of a developer with deep understanding of web security will likely result is a secure implementation. However, at the hands of most developers […] 2.0 is likely to produce insecure implementations.” (Hammer 2012) It also should be noted that OAuth itself does not replace login systems. It is authorization, not authentication, tool that can applications and services use to get access to specific data on user’s behalf. For more complete login system, OpenID would be a better solution. Alternative to OAuth and OpenID could be using SAML (Security Assertion Markup Language) for authorization and authentication.

REST architecture is useful and popular model in World Wide Web. REST stands for Representational State Transfer and, in short, is a way of constructing simple scalable and consistent endpoints for HTTP calls. REST architecture was first introduced by Roy T. Fielding in his doctoral thesis in 2000. Key element to REST is concepts of resource, representation and state. Feng, Shen and Fan describe resources as follows “A resource can be anything. A resource may be a physical object or an abstract concept. If something is important enough to be referenced as a thing itself, it can be exposed as a resource.” (Feng, Shen and Fan 2009). Often resource is something that can be described as a noun. For example, “a user”, “media” or “temperature.” Other main elements in REST listed by Feng et al. include resource identification through UI, uniform interface, manipulation of resources through representations, self-describing messages, stateless interactions and hypermedia as the engine of application state.

There can be two types of states in REST architecture: resource and application state. Resource state holds information about the resource itself and resides on server (Feng, Shen and Fan 2009). For example, a resource “user” has information first and last name “Ted” and “Jones”. This entirety is considered as a state of this particular user entity and should either value, first or last name, change it would be in another state. Application state on the other hand, is about the path user has taken and is located on the client side. Representation is general data about the state and resource (Feng, Shen and Fan 2009).

Web APIs that use REST architecture are called “RESTful APIs”. REST benefits from HTTP methods GET, POST, PUT and DELETE and therefore “restful” implementation is often done using HTTP protocol. When dealing with storing data, there are four general functions known as CRUD standing for Create, Read, Update and Delete. Aforementioned HTTP-methods are used to achieve these basic CRUD actions in RESTful systems. HTTP requests coming to server endpoint is in one form of these methods. Endpoint refers to URL to which REST system responds to. Same URL can lead to different result depending on method of HTTP request. Examples of endpoints (method, endpoint, result, corresponding CRUD action):

- HTTP/GET http://example.com/rest/users/ (list all users, Read)
- HTTP/POST http://example.com/rest/users/ (create new user, Create)
- HTTP/GET http://example.com/rest/users/1234 (get user with id “1234”, Read)

- HTTP/DELETE http://example.com/rest/users/ (remove user, Delete)
- HTTP/PUT http://example.com/rest/users/1234 (update or create new user, Update)

As can be seen from example, achieved result can vary greatly depending on what HTTP method was used. Generally, REST endpoint directs to a collection (e.g. /users) or single element with identifier (e.g. /users/1234). Depending on whether target is a collection or a single element, CRUD operation is done to all items in a collection or to a single element. Generalized relationship between CRUD operations and HTTP requests to a RESTful API is shown in Table 1.

URL	HTTP/GET	HTTP/PUT	HTTP/POST	HTTP/DELETE
Collection (e.g. /users)	**List** all items in a collection	**Update** entire collection	**Create** new element to a collection	**Delete** collec-tion
Element (e.g. /users/1234)	**Return** single element	**Update** element	**-** (not generally used)	**Delete** element in collection

Table 1. Relationship between basic CRUD operations and RESTful HTTP-methods.

PSUP Framework can utilize RESTful API for all interaction between applications and PSUP Storage. Today, RESTful web APIs often return JSON-objects. However, PSUP uses Semantic Web Technologies, and in that field RDF documents are not always in JSON-format, but in formats such as RDF/XML or Turtle. There are two different JSON formats for Semantic documents (JSON-LD and RDF/JSON) so all documents can be converted. However, as different applications may use different formats, RESTful API of PSUP Framework should have possibility to define preferred respond format. This could be done by using query string parameters in requests to RESTful API. Defining response format as Turtle, request URL for a single user element would be then “users/1234?format=turtle”. With proper documentation, developers can use which ever format suits their need best. Newer API language, GraphQL by Facebook has potential for becoming widely used API interface, but according to Facebook developer Lee Byron in an answer to question about potential JSON-LD support, GraphQL has tree-like data structure instead of graph, and therefore is not suitable for Semantic Web JSON-formats (Byron 2016).

2.4.8 Issues with Semantic Technologies

Semantic Web Technologies have challenges yet to be completely solved. In this chapter, research challenges regarding Semantic Web Technologies are listed as explained by Christian Bizer, Tom Heath and Tim Berners-Lee in book Semantic Services, Interoperability and Web Applications: Emerging Concepts (2011). Issues are grouped as follows:

- User Interfaces and Interaction Paradigms
- Application Architectures
- Schema Mapping and Data Fusion
- Link Maintenance
- Licensing
- Trust, Quality and Relevance
- Privacy

(Bizer, Heath and Berners-Lee 2011)

In their dedicated chapter Linked data: The Story So Far, Bizer et al. raise issue of runtime traversal of linked data. While search engines like Semantic Web Client Library (SWCL) has shown that runtime queries can be answered, due to scalability, crawling and caching will be the way of making linked data available for applications fast enough. Other noteworthy research challenge is interface visualization when heterogenous semantic data sources number in thousands or even millions. I.e. how data from varying sources further processed and combined without conflicting entries, and how is this turned into clean representation. The challenge comes from resolving conflicts when different sources provide different values for same property (data fusion).

As all semantic data is represented as URIs, link maintenance becomes issue when RDF links point to outdated or removed data. Bizer et al. do mention that while dead links themselves do not break web architecture, large number of them will cause unnecessary HTTP-requests by clients, which then can be perceived as delays and possibly not existing data for end user. Resolving link maintenance is not an easy task, although few approaches are proposed. Link statuses can be checked and recalculated with scheduled tasks using frameworks like Silk or LinQL or link changes can be tracked by centralized registries with subscription models like Ping the Semantic Web. However, due to decentralized nature of web architecture and semantic web, it is unlikely that issues with dead links can be completely solved. In case of link deprecation, it is clear that data behind that link needs to be either discarded or replaced in practical implementations. Conversely, data quality and relevance are harder to determine. When data is abundant, it is vital to identify most relevant and trustworthy data for a case, especially if that data has not been used or processed before. Data from distinct sources may be of unequal quality, inaccurate, inconsistent, outdated etc. and all of these qualities can also be context specific. That is, one source can be accurate in one domain whereas in another context it may not. In his doctoral dissertation, Hartig proposes trust function that determines context specific trustworthiness score of every Linked Data document or even every RDF triplet (Hartig 2014), which takes trustworthiness of data in the core of Semantic Web Technologies. However, this would also increase complexity of data structures, but could help in data quality issue.

According to Bizer et al. the end goal of Linked Data is to "be able to use the Web like a single global database." While there may be disagreement with that statement, considering Linked Data as one heterogenous and distributed database raises issue with privacy and data protection. Privacy and licensing issues of data will probably require legal means as well as technical, to protect sensitive personal and business essential data.

Semantic Web is based on Open World Assumption (see chapter 2.4.4) and thus it can be said that default state of data is "unknown" unless otherwise defined. That is, like in real life, when faced with unknown sources or terms, they are not automatically disregarded as false. Dealing with unknown states of data entries can be interpreted as semantic vagueness. Moreover, while term definitions should always be available, definitions can be too loose for accurate interpretation or that may be conflicting with previous definitions. Data mappings and Ontology Alignments (see chapter 2.4.5) are used to minimize issues when working with multiple ontologies, but there will always lay level of uncertainty when dealing with Semantic Web Technologies and it causes increased complexity for developers.

Noteworthy issue with Semantic Web Technologies is the slow adoption rate outside academic communities. While Linked Data in general has proven to be beneficial in backend systems of large commercial companies, such as web crawlers and search engine visibility, RDF-structure is not widely used even on popular web frameworks. This could be lack of general knowledge of Semantic Web Technologies among developers. According to two-part survey, which was targeted to experienced developers and project managers, and published in 2011, biggest barriers in applying Semantic Web Technologies are lack of tools, skilled programmers, usability of available tools and standard ontologies. Biggest risks mentioned were exceeding time and cost estimations (Della Valle, Niro and Mancas 2011).

2.4.9 Previous studies

There have been number of previous systems and studies related to personalized or automated user interface creation dating back to 1980s. In conference paper from 1985 "Design alternatives for user interface management systems based on experience with COUSIN", Hayes, Szekely and Lerner provided model based User Interface Management System named COUSIN (Cooperative User Interface) which could be used to build abstracted interface layer on existing applications (Hayes, Szekely and Lerner 1985). COUSIN was structurally model based: all I/O communication between Interface System and application was "slot/value-pairs" where value could be a parameter, command or feedback. Designer would list all required inputs from user and COUSIN would then generate interface for user to fill these "slots" - Practically generating forms for user to fill on the fly. In their paper Haeys et al. about system's control architecture: "A simple print application might have slots for its parameters -- file to print, number of copies etc. A more complex application, such as the file manager […] would have slots for commands and feedback in addition to parameter slots. Since both COUSIN and the application can access and modify slots at any time, the control architecture is mixed." On basic level, UI adaptation was similar to later concepts, that is, UI abstraction layer and generation of forms on demand based on information given system requires.

Jacob Eisenstein and Angel Puerta state in their description for adaptive algorithm for MOBI-D interface development environment, that previous work on automated interface design has had mixed success and that systems developed by researchers are only working on limited domains. As they put it: “However, no technique has been shown to be applicable at a general level. We claim that adaptive, automated systems that solicit and respond to user feedback may be able to succeed where previous efforts have failed.” (Eisenstein and Puerta 2000). As a tool, however, MOBI-D seems to be abandoned.

Other studies include paper by Lavie and Meyer who studied benefits of Adaptive User Interfaces by having multiple levels of interface adaptivity from manual to fully adaptive in driving simulation. (Lavie and Meyer 2010). Their study suggest that intermediate UI adaption should be considered over all-or-none system. This study is examined in chapter 2.2. Gajos, Weld and Wobbrock presented their Supple system for automated personalized interface generation in their paper Automatically Generating Personalized User Interfaces with Supple (Gajos, Weld and Jacob 2010). For interface generation, Supple relies on interface specification, which describes the functionality and purpose of an interface, device model is used to include capabilities and limitations of the platform, and finally usage model called user traces for personalization of generated UI. Gajos et al. approach interface generation from optimization point of view and do demonstrate that despite vast number of possible interfaces, up to 10^{17} as per to their paper, UI generation is still computationally feasible. Supple system does not use Semantic Web Technologies for creation of interface elements. Also, they refer to an automatically generated UIs are often perceived as being less aesthetically pleasing opposed to ones created by humans and that automated systems, such as Supple, should not be considered as complete replacements, but as alternatives to situations where hand crafting UI for every scenario is not feasible or where personalization matters more than aesthetics. Gajos et al. make valid point on practicality of systems like Supple. That is, often interface code is inseparable from application logic and that makes implementation of separate UI system hard. HTML-based web applications, however, are in good position since presentation markup is separate from logic, thus they generally suit better interface generation on the fly. In their paper, they hypothesize of deploying Supple first in the web context as JavaScript library, which is the same reasoning for PSUP to be JavaScript library as well, as will be described in chapter 3. JavaScript and web applications have evolved greatly during the past decade and have moved beyond web browsers to run as native application on various platforms and devices. Therefore, making it ideal platform for UI libraries.

In paper Ontology-based Approach to Adaptation Kadlec and Jelínek put forth the idea of using personal ontologies for UI adaptation. While the paper is merely bringing, in their words, “their attitude to the adaptation process for the Semantic web completely based on ontologies”, it also brings forward some key issues and specifications to be solved before proper implementation. These are:

- ontology retrieval and store service specification
- user profile ontology specification and management policies
- ontology translation service specification
- new concept discovery and classification

(Kadlec and Jelínek 2007)

Kadlek and Jelínek raise the issue of ontology merging (alignment). “Storing user profiles as ontologies necessarily leads to need for an ontology merging process which we can identify as the core of the adaptation process. Nowadays, merging is still evolving – fast and accurate reasoners are needed. To overcome this shortcoming, semi-automatic or human-aided ontology merging is applied.” In their proposal automated reasoning is used to merge information-, lexicon- and user ontologies into one they call “adapted information ontology”. Usage of lexicon ontology is for bringing domain specific vocabulary to ontology merging.

Kanjo, Kawai and Tanaka proposed framework called Adaptation Anywhere & Anytime (“A3”) for user adaptation in their conference paper How to build and adaptive web site: A Framework for User Adaptation (Kanjo, Kawai and Tanaka 2004). They suggest using OWL for storing resources in user profile (user ontology) and XSLT for web sites utilizing A3. XSLT (Extensible Stylesheet Language Transformations) is XML based markup language used for XML transformations in non-destructive way. That is, original XML file remains unchanged, while output is declared in XSLT document similar to how CSS is used to transform HTML documents. Kanjo et al. raise issues with profile inconsistencies and user privacy. Main concept in A3 is similar to what is proposed in PSUP. Their proposed profile manager, UOM (User Ontology Manager), is located within each device instead of being shared from cloud.

In the field of Information Technology, twenty years can be a long time for making predictions. In paper Past, Present and Future of User Interface Software Tools (2000), Myers, Hudson and Pausch predicted future where computers can be mounted on anything (IoT) and the need for variety of reusable UIs. “We are at the dawn of an era where user interfaces are about to break out of the ‘desktop’ box where they have been stuck for the past 15 years. The next millenium will open with an increasing diversity of user interfaces on an increasing diversity of computerized devices.” (Myers, Hudson and Pausch 2000) While their paper concentrates on software development tools, many of their predictions have moved from research topic into commercial products. For example, use of voice recognition in TVs and mobile phones.

3 GUI Personalization Framework with Personal Semantic User Profile

Chapter 2 described what Semantic Web Technologies are and related studies in the field. This chapter outlines the concept and features of Personal Semantic User Profile Framework.

3.1 PSUP Framework

Today, there are huge number of different applications and services trying to attract users. We are bombarded with data and information in many different forms and many applications try to find new innovative ways to apply latest techniques for displaying data better than its competitors. There are many ways to visualize information from plain text to elaborate infographics. It can be said, that visually attractive and logically sound user interface in one crucial element for success of an application or service.

However, attractiveness is individual and even if designers and developers think they are creating enticing way to visualize data, it is not guaranteed that every user is satisfied with it. Visually good-looking data or GUI may not seem so attractive to others because of their cultural background, knowledge, age or for many other reasons. Some people just like to view information in certain way because they are used to it. Moreover, some design choices may impair user experience so much that it may become totally unusable for part of their user base. For example, visual impairs such as color blindness, or bad eyesight can completely prevent using application.

There have been lots of research and many attempts to create more meaningful and personalized user interfaces, including automated and adaptive user interfaces. Some of these are discussed in chapter 2.4.9. However, smart GUIs often aim for effectiveness when adapting to situation or visual fidelity without taking user preferences into account. Also, overriding automated adaptions can often be unnecessarily complicated (Lavie and Meyer 2010). Personalization in GUI should not be confined to visual attractiveness only, but on personalization on more semantic level.

Due to the nature of ontologies, Semantic Technologies are useful for storing metadata and handling hierarchical data. Concepts like “author of this book” or “location of a photo” is fairly natural when thinking in semantic triplets. Semantic Web Technologies has been used for this purpose previously in other theses. For example, in his thesis Panu Vartiainen created RDF engine for automated storing of image metadata in mobile phones (Vartiainen 2003) and Jenni Myllynen applied semantic concepts in genealogy (Myllynen 2007). Ontologies, however, can be used to describe and store more abstract concepts. After all, ontologies are linked data and structurally graphs by nature. The idea for semantic personalization framework in this thesis was proposed by Khiryenko in his paper Semantic UI: Automated Creation of Semantically Personalized User Interface (Khriyenko 2015). This PSUP Framework is based on that.

PSUP is a framework for personalizing graphical user interfaces across different domains, applications and services. It uses Semantic Ontologies for storing user preferences, which makes the profile itself more suitable for world described as Open World Assumption and thus suitable for virtually any application. More on Open World Assumption is described in chapter 2.4.4. PSUP GUI Personalization Framework works as an intermediate between user’s preferences and various services. User can create his/her own semantic vocabulary and store it in his/her PSUP profile in RDF format. Being completely semantic and having no static schema, this profile can hold information in many different forms and vocabularies. Thus, as every person is different from one another so is every user profile. That is, every profile has their own ontology and many same concepts can be told differently from profile to profile. This is the key feature of PSUP: every profile is unique. Other than storing user profile, PSUP will also manage outside modifications to profile with proper rights. Modifications to PSUP are frequent because applications not only use profiles to adjust its GUI or data visualization, but also update it on user’s behalf. See attachments for examples of PSUP basic vocabulary and user ontology.

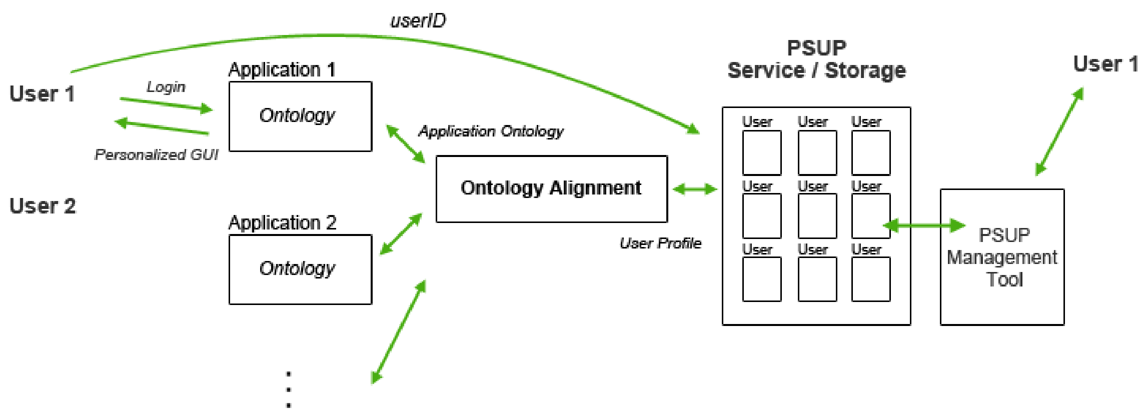


Figure 11. PSUP Framework overview

Today, it is not uncommon to have websites utilize third party profiles for login credentials and fetching some basic info from a user via OAuth. Sites often use common platforms such as Google and Facebook to get user logged in to their services. Google and Facebook, however have solid schema for giving our user properties such as name or email. This profile is not extendable to match information service might need. Using semantic profile allows users and services to store data in format of their choosing. The data in a profile can be something general such as name, profession and location, but it can also be more intriguing such as hobbies, preference on how stock market data is displayed or how long his/her commute distance is; profile can hold anything that may have influence on business logic in an application.

Every Personalized Semantic User Profile is stored in RDF. Since stored data can have entities in multitude of different triplets, applications themselves cannot keep record on every possible predicate. Therefore, applications need to have its own set of ontology in which it defines all the possible subjects that can affect the GUI. When user interacts with the application, (s)he can choose to use his/her profile, which then invokes ontology alignment between the application ontology and user's profile. Alignment is done on other services and can take some time. Therefore, alignment between User Profile and application ontology should be stored in the PSUP storage as well. This saved alignment works as cache making reloading profile much faster in future. Although, should the user profile or application ontology change, alignment will be out of sync and realignment will be required. General data flow with PSUP applications is shown in Figure 11. Users are identified with unique user id. User profile and identification is elaborated in chapter 3.3 PSUP creation.

It should be noted that while PSUP shares some characteristics of an “automated interface technique”, it is not essentially automated user interface system, but instead UI Personalization System. Therefore, actual automated UI adaptation relies on the service utilizing PSUP. Applications themselves can utilize different UI generation systems. However, there could exist library of PSUP UI components which can do the adaptation automatically. Reusable UI Components is described in chapter 3.6.

PSUP as a concept could work platform independently, but for this thesis, focus is on how it would be implemented in websites. Normally websites use wide range of different JavaScript libraries, frameworks and tools such as jQuery, Backbone, React and Vue. Implementing PSUP framework to a site should be as simple as possible, thus best scenario would be to create it as a JavaScript library too. Preferably one made with native JavaScript with no third-party dependencies such as jQuery for future proof implementation. From website maintainer’s perspective PSUP JavaScript library would be just one drop-in among the other libraries. Of course, if the site is not previously built from compatible elements or in general PSUP in mind, it will require a bit more work from the developer. Implementation to a website is described in further detail in chapter 3.5.

Being semantic and schema-free, PSUP system can be used in heterogenous system environments for transferring user related data between machines or physical devices. Personal Semantic User Profile is not limited to Graphical User Interfaces can be utilized in any Interface that can benefit knowing user’s preferences or ways of doing things. For example, PSUP can be used in IoT environment where devices can come from multiple different vendors and they do not necessary share common API. That may not involve user at all, but instead interfaces are for machines. In general, devices can interact and share data automatically with each other if they have ontologies to match with. However, machine to machine interfaces and web services are not only interfaces related to personal semantic profile. Natural User Interface (NUI) is another system that can utilize semantic profiles for personalization. NUI is term used to describe interfaces that are unobtrusive or even invisible to a user, and they are easy to internalize, such as some touch or virtual reality interfaces, or using voice to control computer system. For example, in virtual environment personalization can be used to fine tune motions user is required to do as one user can prefer wider arches of movement, whereas other user is less exaggerative.

3.2 PSUP Framework structure

In this proposed GUI Personalization Framework, there are three main “players”: a user, PSUP Service and an application or service implementing the PSUP. This relationship is shown in Figure 12. Common use scenario would be as follows: user launches an application or website and (s)he is then prompted to login with his/her PSUP credentials, if not already logged in. Website then fetches user’s profile from PSUP Storage using PSUP API, aligns the profile ontology with its own and adapts the user interface as best as it can to best serve the users preferences. If user is already logged in, this is done automatically in the background. If user’s profile is already aligned to applications ontology, PSUP API returns stored alignment. User can change the interface within a website and these changes are then updated back to the actual profile. These steps are elaborated in following chapters. API implementation is open, but in this thesis it is considered to be in RESTful architecture using HTTP. Other protocols, such as SOAP, are possible as long as the API does not need to worry about application nor profile states. API requests are not publicly available. That is, CRUD operations on user’s behalf require valid access token to be included within all requests. This is to prevent malicious access to user data. Ontology alignment is handled separately with Java Alignment API and Alignment Server (Euzenat 2016) based on The Alignment API 4.0 (David, et al. 2011).

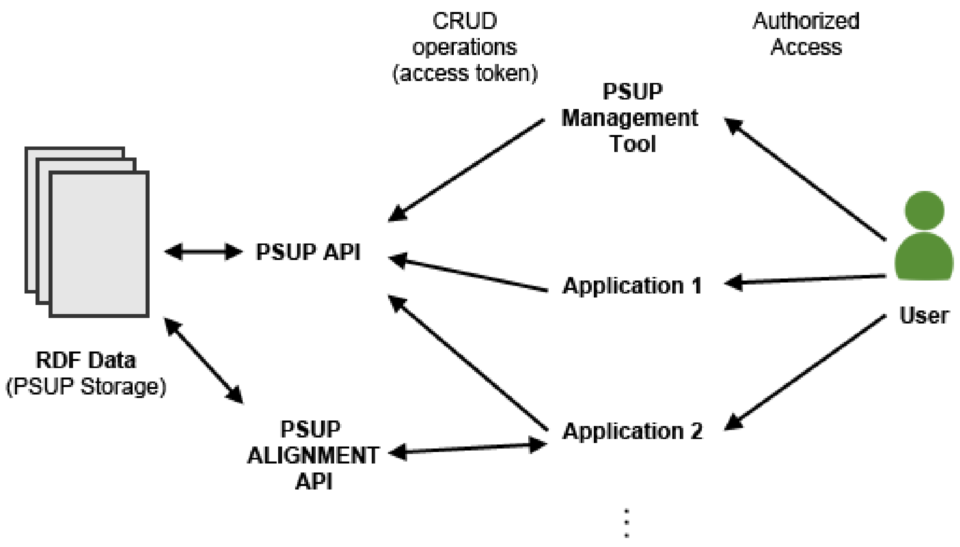


Figure 12. User, PSUP and applications relationship.

As PSUP is separate and independent from systems that implement it, PSUP Service offers interface for users to create, view and modify their profile. This management tool can be the start point for creating profile but since one cannot create complete profile of one's life at once, most of the profile would be filled as the user goes on using services and applications. To be more specific, best case would be when user would not have to specifically edit his/her profile at all. Instead, PSUP would always adapt based on user's browsing habits and decisions (s)he makes when creating or consuming data. User might never need to use management tool because (s)he can create profile without leaving a site or application. When user goes to a website that is utilizing PSUP Framework, user is offered to create profile on the fly. More on PSUP profile creation on chapter 3.3. However, there may become a situation where user needs to change something specific and therefore an interface for direct profile manipulation is good to exist. Also, user may have general interest or privacy concerns about what kind of data is stored in the profile, or user wants to download all his/her profile data for offline use. Raw Turtle notation used in PSUP RDF files is somewhat human readable, but might require more technical knowledge for better understanding. Therefore, management tool requires better visualization for it to be usable for everyone.

As the world gets more and more connected, functioning Internet connection, albeit not reliable all the time, could be assumed. Therefore, main proposal for storing Personal Semantic User Profile is in the cloud, but for connection problems or offline use by user's choice PSUP could be stored locally as well. When user goes back online, PSUP is then synced back to the cloud. In future, PSUP could be used on local application and even on operating system level. Local profile, however, brings a range of technical issues, such as actual location on different platforms, and update order if same properties have been updated in offline on separate devices. Thus, in scope of this thesis, PSUP storage is assumed to be located on online servers (cloud). Also, for the scope of implementation being by JavaScript library, automatic adaptation on unvisited sites is not possible. Login on single site can be stored in cookies, but as cookies cannot be cross-domain, user will not be automatically logged in when (s)he arrives to a new site. This will mean that user does not get automatically personalized GUI, but instead (s)he must wait for the JavaScript library to load and then login. If user profile were part of web browser, e.g. via extension, PSUP could be utilized on page render time. This would result in better user experience.

3.3 PSUP creation and structure

For Personal Semantic User Profile to be popular, it needs to be as automated and inconspicuous as possible. If registering takes too much time, conversion rate, that is how many users who start the registration process will finish filling in the form, is going to be low. Users generally do not have patience to go through forms spanning multiple pages setting up info they might not need. Ideal option is to have PSUP creation possible on every website that utilizes PSUP JavaScript library. Having popups is considered distracting, but websites could have other non-disruptive ways to inform user that (s)he can customize his/her user experience. For example, simple bar on top of the site or message appearing in one of the corners.

For user to be identified, only email is required. Even password can be left out as, given that valid email is received, system can send expiring single use password for the user. User could be prompted to give password later, but (s)he could also continue receiving one time codes every login. Single use verification codes can be used alongside normal password for extended security. So called two-factor authentication requires user to enter verification code (s)he receives as email or, if phone number is provided, as SMS after entering normal login credentials. This way user account is safe even if someone else gets access to its username and password.

PSUP login identification is email, optional password (if user doesn't want to use one-time tokens), optional phone number for two-factor authentication and optional real name field. There shouldn't be too much required information. For being as flexible as Semantic Technologies are, users should have the power to choose what information is stored. Some might want to use separate profiles for work and free time. User's login credentials are held separately from actual semantic RDF-data for example in SQL-database. User's common information can also have other details as well, although it is not necessary for PSUP to work. This additional information could include usage logs for possible error situations, or fallback details for recovering lost credentials.

As seen in Figure 13, there are at least two possible ways to arrange user's RDF-data: all the semantic data is in one big pool of data, or application specific semantic data is stored independently, but still having some shared common data. On higher abstraction level, single profile entity may seem clearer, but considering implementation, having multiple sub-profiles have following benefits.

- Better separation of concerns
- Security, single applications cannot see data given to others
- Easier to maintain as every sub-profile can be stored separately (allowing e.g. partial offline profile)
- Easier to avoid namespace issues and thus prevent possible security holes as namespaces and prefixes can be faked

As a downside, applications may want to access similar data and this will lead to data duplication. User can have same preferences multiple times in his/her Semantic Profile. To avoid data duplication, user can have one shared RDF that is visible to every application. This shared ontology holds most common user data such as general likes/dislikes, major disabilities etc. When application then tries to match user profile against its own ontology, PSUP could first use this shared space to find matches and, only for properties that are not found, offer user to add new information either in shared space or in application space within his/her Personal Semantic User Profile.

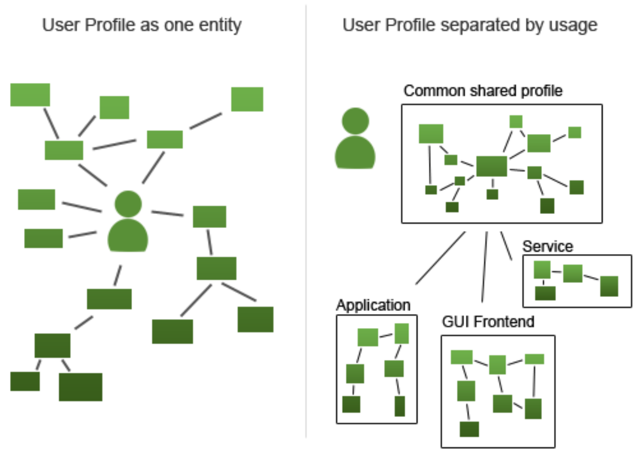


Figure 13. Two ways for storing profile data in PSUP

Every User Profile is initially created by using PSUP Ontology which defines basic vocabulary and concepts of a profile. These concepts include common properties that could affect business logic of various applications, such as “profile” (top level class), “preference”, “likes”, “dislikes”, “hasRestriction”, “age”, “location” and “isInterestedIn”. Root element of every Semantic User Profile is an instance of Profile class. Naturally all OWL, RDF and RDFS properties are also available and applications can have their own vocabulary.

It is clear that one ontology simply cannot cover every possible concept that can affect GUIs. Therefore, every user is provided with possibility of creating own ontology on top of basic PSUP vocabulary. User is free to create his/her own objects and predicates or use many open vocabularies (see chapter 2.4.3 Open vocabularies). However, creating own vocabulary can be daunting task for a person who does not have knowledge in the area of Semantic Web unless (s)he is properly guided. Best case would be if GUI for creating ontology is very intuitive to use, thus, average user does not even need to know what Semantic Web is. Interface should allow directly writing simple sentences, such as “I read lots of books” for user to describe themselves. Then, using Natural Language Understanding (NLU) methods, text is parsed into semantic triplets that are stored within user’s PSUP. NLU is considered AI-Hard (also known as AI-Complete) problem (Yampolskiy 2012) and modelling of the semantics of adjectives is also considered difficult task (McCrae, et al. 2014), but using restrictions in sentences, like length and language, would help implementing such system to PSUP Framework.

3.4 Updating PSUP

One of the key features of Personalized Semantic User Profile is that it evolves over time. More accurate profile means better user experience with services user has used in the past, and with services user is yet to experience. There are multiple ways on how PSUP gets updated. These methods can be roughly divided into two categories: user originated and automated changes. User originated methods include all PSUP update actions where user makes the decision whether in normal interaction with different GUIs or directly modifying his/her profile with PSUP Management Tool GUI, that is, manually update PSUP records. Automated update events happen without user's direct involvement when GUIs align to user's profile. Applications may also make updates to user's Semantic Profile, but it should be clear for the user when this happens so that user can ensure his/her PSUP still represents himself/herself.

3.4.1 Ontology alignment and automated update methods

When a user lands on a website or uses an application, (s)he can log in to PSUP Service via OAuth methods, and initial ontology alignment between his/her PSUP and application's ontology is generated. Application can use matching information to modify user's experience by changing its GUI or business logic per user preferences. From application's point of view, this is what there needs to be for PSUP implementation. If application uses just PSUP Service API, steps would be as follows:

1. Application makes authorized API request to PSUP Service for user data given that user has logged in or otherwise granted permission with valid access token. Application needs to provide its own ontology with the request.
2. PSUP Framework generates ontology alignment either internally or using separate alignment service.

3. Alignment summary is sent back to application, where user must verify matched items. User may also fix mismatches, provide missing information or remove matches (s)he is not comfortable with.
4. Applications sends user confirmation to PSUP Service.
5. PSUP Service responds with aligned data values.
6. Application can now adapt according to user profile. Providing service developers have built various GUI and logic behaviors.

This ensures that 1) user data is protected as no data values are sent until alignment is accepted, 2) user can provide only data values (s)he feels comfortable with, 3) user can easily add missing data, 4) no additional ontology data needs to be sent between PSUP Service and application and 5) every alignment is sent back to PSUP framework for future use. This will give best security for user data. Alternative and more distributed version would be to allow services use third party alignment services. However, if user profile is structured as one big entity (see chapter 3.3), possibly malicious application can receive all user profile entity names. Although entity values are not explicitly provided without user permission. Getting access to entity names alone may still cause privacy concerns. Data flow would be as follows.

1. Application makes authorized request for user profile but do not provide its own ontology.
2. Optional: User is prompted to authorize access to full profile entities.
3. PSUP Service needs to send either shared or full profile entities depending on his/her PSUP data structure. No data values are sent.
4. Service sends ontologies to third party alignment service or has its own methods.
5. After ontology alignment, applications must send it to PSUP Service for actual data values according to matched entity names.
7. PSUP Service sends alignment summary for user to accept. User can modify, remove, or provide information on missing items.
8. Applications sends user confirmation to PSUP Service.
9. PSUP Service sends aligned data
10. Application can now adapt according to user profile.

Services utilizing PSUP may have its own methods for alignment or it can distribute the work to third party alignments services. Third party alignment services may be preferred for better performance or accuracy. However, if alignment or other parts of implementation is distributed to third party services, more steps are required to ensure security and proper data flow. PSUP Framework suggested in thesis this does not require alignment to be done via PSUP API but it should provide at least basic level alignment API for simple use cases. As mentioned in chapter 3.1, regardless of ontology alignment service used, alignments should be sent to PSUP Service to be stored for later use. This is because alignment process can be time and resource consuming, therefore using cached version saves time and bandwidth.

Basic dataflow of initial ontology alignment was stated above. However, this should be considered as a starting point for automatic PSUP evolution. PSUP updates can flow bidirectionally between PSUP Service and applications implementing it. This means that applications can update users profile automatically. Actions made within services and applications echo back to user's profile, thus making updates to user's PSUP on the fly. Depending on user's actions and repeating behaviors, application can use its own vocabulary to make updates to user's PSUP. Thus, making application specific personalization more accurate or adaptable should user's preferences change over time. Reasons and logic for updates is in the hands of application developers, not PSUP Framework. For example, if user who browses parts for a motorbike, store application may deduce that user owns one, is interested in motorbikes and is generally interested in motor vehicles and therefore make update to that user's PSUP as well. However, in domain of online shopping, store applications gather lots of information on its users and may also base other deductions from other user behaviors as well. For example, based on user behavior statistics, users that browse items in certain category are usually also interested in products on another category as well. Assumptions based on statistics are not true for every customer and therefore should not be a justification for Semantic Profile update. Personal Semantic User Profile is based on facts that user can agree to, thus keywords and triplets in semantic ontologies should avoid modal verbs that indicate uncertainty, for example predicate "mightBeInterested". To make sure user can agree to his/her profile, all updates should be informed to user unless (s)he has explicitly given permission to an application to modify his/her profile in background. PSUP JavaScript should include functionality of showing message when web-sites want to make updates on his/her profile. For other platforms, such as native mobile applications, PSUP API requires new token for important profile updates and thus, pre-venting silent updates without user's acknowledgement.

As a person uses different applications, his/her PSUP will grow. At some point the infor-mation new application requires is very likely already within user's PSUP. In this case, it would be beneficial for new application to have access to previous data used in similar applications. However, depending on data structure this may be hard. In previous chapter, two different structures of user ontology were suggested: full profile as one entity or multi-ple sub-profiles for each application. If PSUP structure is stored and handled as one big RDF storage, it will cause privacy concerns more easily as every automatic alignment is done directly against full profile. This is the case if applications are using third party ontol-ogy alignment services. Naturally user could be required to find all matching pairs between his/her PSUP and application ontology, but considering the possible size of ontologies, it is most likely far too big task for any person. Therefore, alignment should be done automati-cally as far as possible and only involve user in situations where term match was not found. For accurate automatic ontology alignment to be possible, align service must be provided with all (domain) dependent information.

As suggested in previous chapter, better solution compared to one ontology RDF would be to divide user ontology into separate subsections where most commonly used vocabularies and preferences are stored in shared ontology and application specific vocabularies are in their own. However, if PSUP structure is divided into separate sandboxed sections, it can cause unnecessary data duplication as mentioned in chapter above. It will also add com-plexity to RDF storage and data sharing between applications, because applications do not have direct access to data in another section. Technically application specific ontology could be in separate RDF file or database, but PSUP Service should contain mediator which controls all incoming queries. That is, full ontology is not provided directly upon requests, but this mediator will query and parse required entities from both shared common and application specific sections of user's PSUP, thus making it possible for all user data to physically locate in same RDF file or database. Having two separate sections within PSUP RDF will raise questions on what grounds properties are divided between general and application specific sections. In general, this should be left for the user to decide. During ontology alignment summary, user could have possibility to choose URI prefix of any given ontology entity. It is acknowledged, however, that some data duplication may be unavoidable.

3.4.2 User originated update methods

All methods where user manually makes changes to visualizations and other GUI behaviors can be considered as user originated updates. This can happen when user directly changes the GUI while using it. When presented with data, (s)he might be given a choice on how it will be displayed assuming application or website has implemented multiple visualization styles. Choice might be simple a checkbox, dropdown or more elaborate popup with detailed instructions and preview of how data will look on each choice. This gives control to user, although, (s)he will most likely find it bothersome should (s)he be forced to make all choices at once upon arriving to a site for the first time before (s)he can get access to the data (s)he originally came for. Therefore, in most cases it would be better if services set default values for data visualization and other personalize values. Websites may even choose to not use personalization framework site wide at all. Nothing prohibits websites utilizing PSUP on just distinct sections of a site. Using default values would also make transition to personalized web less obtrusive to current browsing experience, because personalization works as “opt in”, that is, user can adopt it if (s)he wants to, but everything still works as before on default.

Given the scope of this thesis, semantic personalization can only work on sites that use PSUP JavaScript library. Easiest way to make website element PSUP compatible is to explicitly initialize it with PSUP JavaScript methods. Actual technical implementation is detailed in chapter 3.5. In simplest implementation style, it is up to website developers to make personalization options available for each GUI component. However, for more complete web wide personalization, web developers should be offered tools for implementing PSUP with pre-made GUI widgets that automatically utilize semantic profile. This saves time and cost of making adapting GUIs available to users. PSUP widgets and functionalities are proposed in further detail in chapter 3.6. Every PSUP widget can be individually modified from options icon on that element. For the sake of usability and avoiding clutter in UI, it is better to hide those icons on default. Instead, message or icon is shown on top of the website from which user can toggle highlighting of all modifiable items. Many browsers support third party extension and should PSUP be one, PSUP icon could be shown directly in browser toolbar as well and thus freeing space for visualizations. Possible locations for PSUP Modification indicators are shown in Figure 14. Another reason for showing PSUP icon or message on top or in browser toolbar is for giving the possibility to make site wide alterations, such as currency, font size etc. that will affect all GUI widgets used on the site.

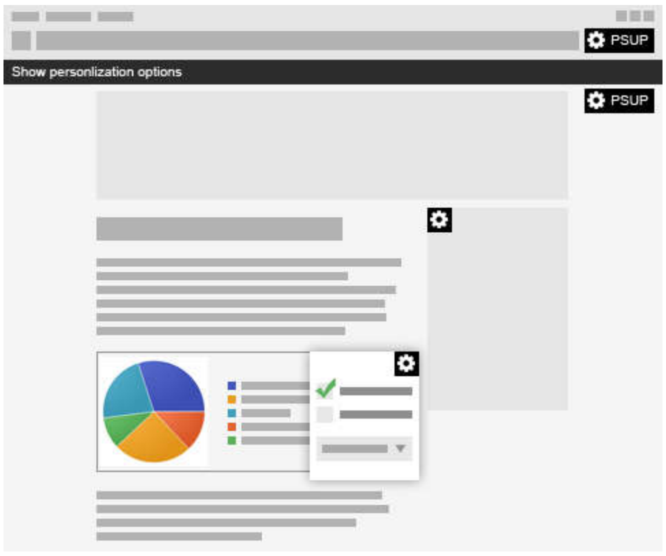


Figure 14. Possible locations for PSUP modification indicators within a web browser toolbar or web site. Modifications can be site wide or distinct element specific.

Over time user's semantic profile will grow allowing more complex interactions and automatic deductions of preferences (s)he might not have manually given (derived information) and thus, giving more accurate profile of user as a person. However, overlapping semantic entities within user ontology might cause conflicts. Conflicts may be caused by user errors, but also changes in state of mind. Personal preferences are not the only factor in user behavior. Among cultural background and cognitive skills, user's mood, motivation and goals can change during a day (Lavie and Meyer 2010). Storing and changing situational preferences is challenging due to inability of detecting changes in user environment. Location dependent preferences could be read via APIs device or operating systems offer, but location alone may not be the deciding element for changing GUI or preferred input method. More specific measures are needed to detect if a person is moving via bike (prefer voice commands) or tram (use bigger icons). However, detection of environment changes is not in the scope of this thesis. Situational preferences also raise a question of how it will be stored. If a user makes direct changes in PSUP widget, it is not clear whether (s)he prefers it always like that, or just for this one case. If user PSUP is not large, preferences can be approximated either by using only the best matching (see chapter 2.4.5), last or averaged numerical value. With enough data, user behavior can be calculated more precisely with machine learning methods.

When UI is changed in runtime, PSUP is updated on the fly via AJAX request to PSUP server. Preferably the data sent to PSUP server does not hold specific information about what visualization was selected for a part of a site or data, but instead semantic information of what type of visualization is preferred. The information stored to PSUP should be semantically general so it can be applied in other sites as well. For example, if the user wants key features in audio related products to be shown in table format, triple stored to PSUP would be:

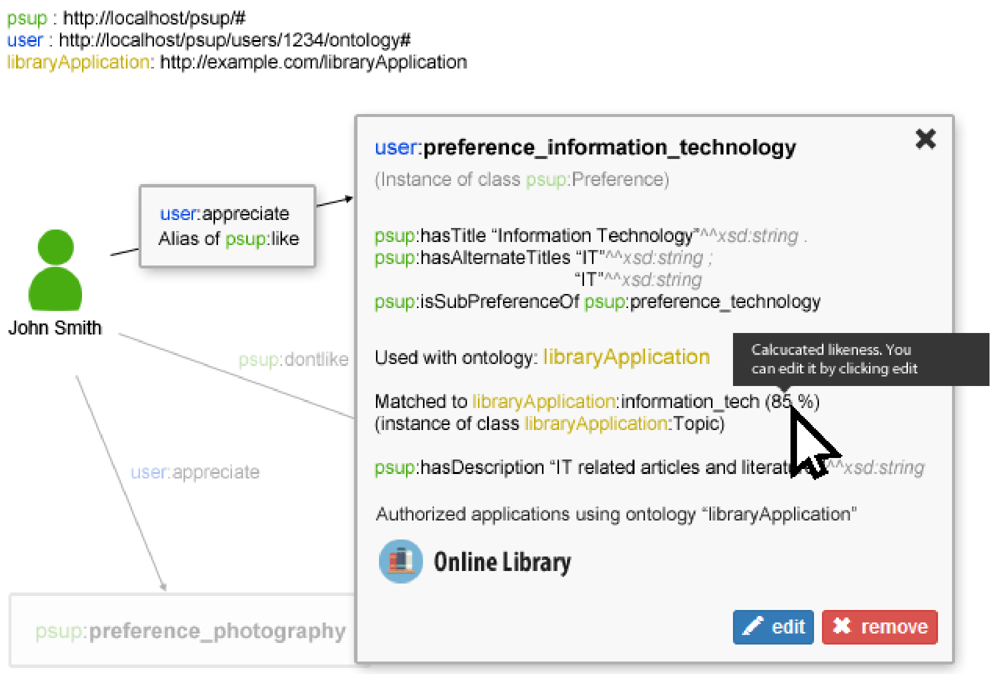
user1234:audioProducts psup:preferredDisplayFormat "table" .

Subject *audioProduct* refers to an entity in user's ontology. This triple resource would be stored either in user's general ontology or application/site specific sub ontology (see chapter 3.3 PSUP creation). Technically, PSUP does not prevent storing single, specific choices such as "for this product only, show data in table format", but should product be removed, that resource will become meaningless within the profile unless PSUP automatic cleaning methods are developed. Redundant data instances can be manually removed with PSUP Management tool, but as PSUP is likely large data store, it should not be responsibility of a user. Therefore, if specific choices are made, they could be time limited and removed after expiration.

3.4.3 PSUP Management Tool

PSUP Management Tool is a separate interface for viewing and modifying personal profile. Management Tool allows user not only to edit personal information directly, but also him/her to see all the applications (s)he has given permission to view and edit his/her PSUP. Management tool is available online from a server and requires user login credentials described in chapter 3.3. Technically PSUP Management Tool acts very much like every other service or application using PSUP Framework. That is, it can use the same PSUP API as other GUIs.

As PSUP can grow very large, data visualization becomes more important, especially for non-technology oriented people. Advanced user can understand RDF notations and they can be offered an access to edit RDF-data directly. However, even for advanced users, this can be error-prone without proper code highlighting and syntax spellchecking, thus, GUI for editing RDF data is needed. People who do not have knowledge of underlying technologies can see relationship between different data entities if they are represented in graphical format. Thus, one natural way of visualizing RDF data is graphs. Yet, if profile data set is big, it can be hard to get full picture from a simple graph. Also, user may want to know to what parts of his/her profile a service has access, why, and how service uses that information in its GUI. If user removes a node, changes its identifier or value, how will it affect user experience? An example of a view in management tool UI can be seen in Figure 15. In this use case, user has opened a node which represents general interest in information technology. Applications could use this information to pre-filter content that is in category of IT, for example, in online stores, news or library content. In figure, blue prefixes are used to indicate user vocabulary, thus, “preference_information_technology” is instance of preference class, which is described in PSUP basic vocabulary (see chapter 3.3). Basic vocabularies are prefixed with green “psup:”. Other useful information includes key details of preference instance such as title, alternative titles and data hierarchy. Ontology match for a library ontology is shown including likeness to the matching element within library ontology (85%). Vocabulary from library application is shown with yellow prefix. Applications using this instance are listed for quick recognition of where current item is used. User also has ability edit and remove the instance.



psup : http://localhost/psup/#
user : http://localhost/psup/users/1234/ontology#
libraryApplication: http://example.com/libraryApplication
John Smith
user:appreciate
Alias of psup:like
psup:dontlike
user:appreciate
psup:preference_photography
Online Library
edit
remove

user:preference_information_technology
(Instance of class psup:Preference)

psup:hasTitle "Information Technology"^^xsd:string .
psup:hasAlternateTitles "IT"^^xsd:string ;
"IT"^^xsd:string
psup:isSubPreferenceOf psup:preference_technology

Used with ontology: libraryApplication

Calcucated likeness. You can edit it by clicking edit

Matched to libraryApplication:information_tech (85 %)
(instance of class libraryApplication:Topic)

psup:hasDescription "IT related articles and literat…"^^xsd:string

Authorized applications using ontology "libraryApplication"

Figure 15. UI visualization for PSUP Management Tool

Management tool requires clever ways to filter, search and display relevant data instances for the user. Semantic technologies make keyword based search more versatile as search results can show similar keywords and synonyms used in different application ontologies. That is, user who is keen to see his/her stored interests likely uses keyword "interest" in search field. However, an application might use keyword "hobbies" for reading similar information in its personalization options. Given that ontology alignment is done between user and application ontology, some type of similarity match is likely been made for both keywords, thus, search result can intelligently show matches from application as well even if keywords are not similar with string based techniques. With intelligent automation, PSUP data could be reduced and combined, and redundant instances could be detected and removed. However, design of automated data reduction is not in scope of this thesis. Building easy-to-use interface for controlling RDF profile, which can potentially contain life span of user information and preferences, is a vast task and requires studies on its own.

3.5 Implementation

As described before, PSUP framework is built on Semantic Web Technologies. Data is stored in RDF storage, files or database, and requests are done by using Simple Semantic Web Architecture and Protocol (SSWAP). Alignment process is described in further detail in chapter 3.4. In short, valid user access token is required when requesting user profile and alignment is done by either PSUP or a third-party service. In this chapter requests and other implementation details are elaborated.

Today, websites can often be considered as web applications due to lack of traditional document structure: instead of loading full documents, web applications often generate new views or update content partially on the fly via WebSocket protocol or AJAX requests. Not only web applications, but distinct application UI components can be examined as separate web services that individually attend to their state and content as well. Thus, personalization can also be done on a whole application level or on single UI component level. Each component, upon receiving user data, can modify output of its HTML code regardless of application or other component states. When page is initially loaded PSUP JavaScript library detects all uninitialized PSUP compatible UI components. Request(s) for user data from PSUP API is then initialized and after ontology matching process, UI elements are personalized according to the user data. If no match for any data entry is found, user is prompted to fill it in and PSUP will be notified of it (insert/update) as described in chapter 3.4. User has also an option to ignore missing data request, which would lead to UI component to be in its default state. PSUP request can also be explicitly triggered e.g. upon receiving new data from a server, or by user action. Steps for receiving user data for UI component personalization is visualized in Figure 16.

In HTML based user interface, UI component capable of PSUP based personalization can be marked with descriptive class name that PSUP JavaScript library can detect upon page load. However, since HTML5 allows the use of data-attributes, those could be used as well for marking PSUP capable elements and storing values within HTML element itself. This not only reduces the need for keeping track of additional data structures on front end, but also makes distinction between application logic and rendering clearer, since class-attributes are often associated with Cascading Style Sheets (CSS), and respects HTML standard.

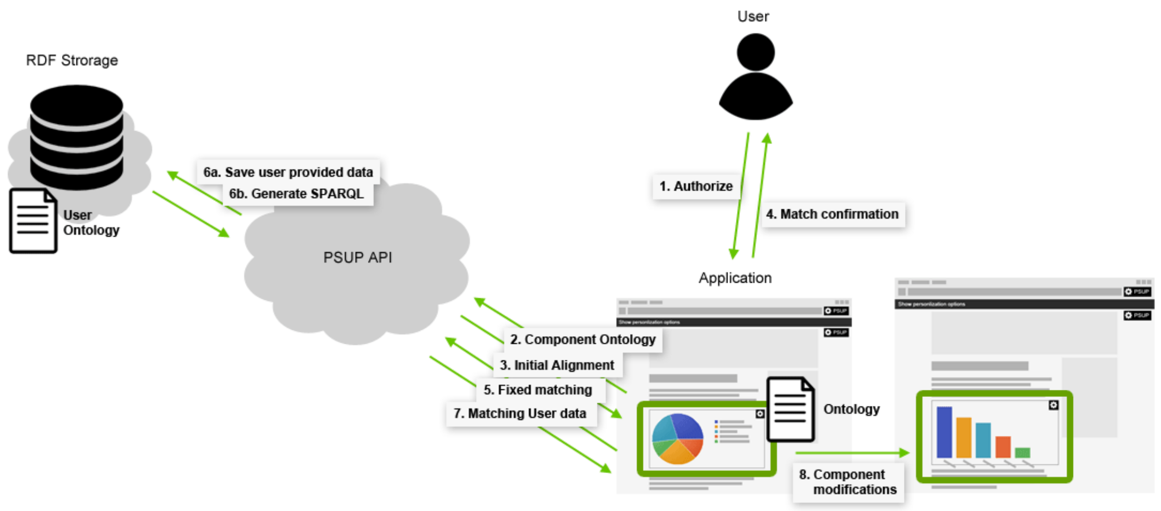
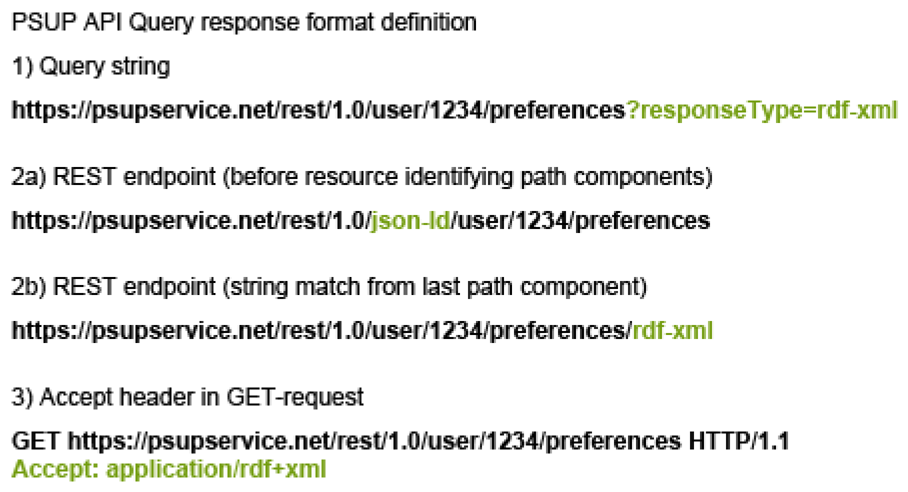


Figure 16. Data flow of UI Component personalization

Upon receiving data request, PSUP generates applicable SPARQL query against stored data within RDF storage and returns it to application or single UI component. It should be noted that, for implementation interoperability between different platforms, PSUP service does not return application rendering specific markup. For instance HTML as not all applications are HTML based. Many applications for specific, often closed, systems use their own proprietary systems for view rendering. Format of data that application receives can be determined in the request, but semantic formats such as RDF, RDF/XML, JSON-LD or RDF/JSON are preferred. This way, implementation of personalization effects is left for application developers or UI components. Considering PSUP API being RESTful architecture, request for user data can be done via HTTP GET method. As seen in Figure 17, desired response type could be defined in query string in requested URI (as described in chapter 2.4.7), within RESTful endpoint URI itself or by Accept header in GET-request. Accept header lists the MIME types of a requested resource client system is willing to receive. The benefit of this is that it allows definition of multiple response types in one request as accept header is a comma separated and does not require modifications to API REST endpoint structure, thus would be considered as best practice. However, all given options are a matter of endpoint configuration and therefore possible. Albeit, PSUP Service should only accept one to avoid confusion and implementation of the same functionality multiple times.



PSUP API Query response format definition

1) Query string

https://psupservice.net/rest/1.0/user/1234/preferences?responseType=rdf-xml

2a) REST endpoint (before resource identifying path components)

https://psupservice.net/rest/1.0/json-ld/user/1234/preferences

2b) REST endpoint (string match from last path component)

https://psupservice.net/rest/1.0/user/1234/preferences/rdf-xml

3) Accept header in GET-request

```
GET https://psupservice.net/rest/1.0/user/1234/preferences HTTP/1.1
Accept: application/rdf+xml
```

Figure 17. Methods of defining API request response type

To keep track of applications using person’s semantic profile, applications could be required to register themselves to PSUP system. This helps identifying requesting applications, permissions they have for person’s data and allows them to describe reasons why certain data is required. From purely technical stand point, application registration is not mandatory, but in practice it will help keeping requests in order. Upon registering in PSUP management tool, every application receives identification key (string hash) which can be used (or required) in every API request. Registering allows developers to inform PSUP Framework different domains and namespaces in which their application operates on. This gives semantical context. Even further, applications could be allowed to preload their ontologies (just vocabularies, not entire data sets) to PSUP for pre-calculated (reasoned) ontology matching against most common predicates. Also, in case of website, informing domain name prevents malicious use of hijacked identification key. That is, requests using an identification key is only allowed from set of domains and subdomains defined by application developers and malicious use is easier to track down.

Access token generation is straightforward process. Access tokens are used to verify application's privilege to user's information. That is, user has been informed and by his/her action application is authorized to his/her full or partial semantic profile, or permission to modify it. User action can be as simple as a simple click of a button if intent is clear. General data flow of API request was shown in Figure 16. In Figure 18 first step (authorization) is elaborated. For security reasons, permission request is generated by PSUP service either by redirecting user to PSUP domain or via popup or a separate browser window, thus preventing faux permission views. This way user can be sure that the privileges that the application is requesting are real. Otherwise, malicious application could fake permission list and user might be fooled to give access to information (s)he is unaware of. For every request against PSUP API, valid access token is required in request HTTP header. Should the application require additional information, i.e. outside the scope of existing access token, new access token is required and user is again prompted to accept it.

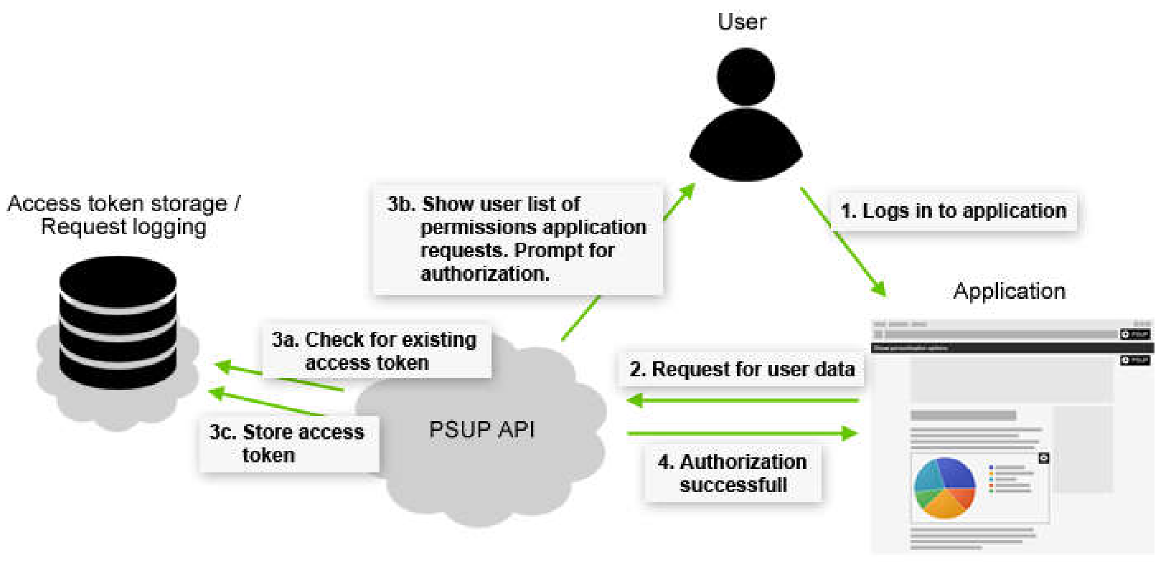

User
Access token storage / Request logging
3b. Show user list of permissions application requests. Prompt for authorization.
1. Logs in to application
Application
3a. Check for existing access token
2. Request for user data
3c. Store access token
PSUP API
4. Authorization successfull

Figure 18. Granting access to user information.

Keeping track of requests and access token requires a storage system. A simple database system is suggested for that. As every application has identification hash (as mentioned above), it can be used to index every request by that application. Logging database is part of PSUP system, however, physically it can reside on a separate server. Logs can be used to detect malicious actions, analytics or debugging. Similar to request logs, access tokens are stored in database with information such as generated token, user id, application id, data privileges given to this access token, expiration time and possibly device or browser identification should permission be limited to a single device.

It is clear that PSUP Framework service requires active maintenance, thus raising the question of development, administration and upkeep of the entire system. For active development, open source model is suggested. Core team would be responsible for acceptance of pull requests and new features from contributors and community. For distribution, PSUP system could be considered as a type of "as-a-service" system instead of a single monolithic service or product maintained by single organization and residing in one server. That is, PSUP Framework is downloadable from dedicated open source repository and organizations can then proceed to install PSUP System of their own, thus, creating a network of PSUP systems. With discovery system via SSWAP, separate PSUP Frameworks can be found. User might be more inclined to use GUI personalization system if it is maintained by an organization (s)he trusts. Skillful user could even create his own private PSUP repository. Naturally, maintaining discovery of many PUSP providers can be hard: databases may not be current, some entries might not be available anymore and some might be yet to discover. Some PSUP services might be deliberately hidden if they are private. Therefore, on PSUP login user could be shown a list of most common PSUP services, but also text input where (s)he can write URL of some other PSUP service. From application's perspective it does not matter where user data comes from as long as it is compatible with alignment service.

3.6 Reusable Semantic UI Widgets

UI adaptation between an application's and user's ontologies was described above. Despite of issues of semantic web technologies mentioned in chapter 2.4.8, using semantic technologies for GUI adaptation in PSUP Framework is still a quite straightforward concept. That is, application has a set of vocabulary which affects its UI rendering and logic. Upon user arrival or interaction, this vocabulary is compared to user's profile for similarities. If user data has matches, or user creates them on the fly, application UI can be adapted according to matched items, which effectively describe user preferences. Although this leads to more interoperable user experience from application to application, it is still limited to time and resource restrictions during the development. This is because developers should make UI adaptable to their own semantic vocabulary, meaning they are required to implement all possible UI visualization and logic models themselves. Instead of making one UI, developers need to make several UI permutations from different parameters, and that requires more development time, and therefore more money and resources. Naturally companies are free to decide which parts of the UI are modifiable by user preferences, thus they can choose best resource allocation for most competitive final product. Even if semantic personalization only works on few key features of an application, those can make clear significance against competition. However, while PSUP Framework is ubiquitous due to its ability of being implementable for different platforms, it is still dependent upon how much companies can gain business value from time, money and other resources they need to invest. Every new feature and visualization option in application logic makes application state management more complex, which can lead to coding errors and security attack vectors within product. This eventually leads to lowered user satisfaction. Considering these reasons, reusable, semantic and PSUP compatible widgets are required.

In his paper, Semantic UI: Automated Creation of Semantically Personalized User Interface, Khiryenko proposes semantic API for discovering visualization modules based on their functionality, that is, inputs and outputs, thereby allowing developers to build semantic interfaces block by block. Using SSWAP for service-to-service communication and Resource Description Graph (RDG) for module descriptions, interfaces can be built automatically based on application ontology, but also taking user preferences into account (Khriyenko 2015). This means that application's GUI can be partially or even entirely outsourced to semantic widgets that are loaded on-the-fly as user loads a page or application, thus, replacing static and immutable UI components. Developers' only control is determining what inputs and outputs are allowed from third party widgets so that they can store required business relevant information on their own systems. Benefit of reusable widgets is that, assuming suitable widgets are available, GUIs become truly personalized as every user can be provided with, not only unique, but also preferred set of GUI components according to his/her semantic profile. Also, instead of using resources on GUI components and visualization, developers have more time to work on other parts of their application, such as business logic. If widgets become popular and numerous, entire application could be built based on those. However, there are multiple downsides as well. For truly unique user experience, there needs to be lots of widget providers as combinations of inputs and outputs, that is different use cases, are huge. Being semantic in nature, widgets have benefit of not having to be exact match for application's requirement unless that requirement is critical for business logic. This is because ontology matching is determined by similarity values (see chapter 2.4.5) and if output from a widget is close enough for application, it is usable for a given GUI component location. This is the case in simple visualization preferences. However, when dealing with user identifying information that is critical for business logic, such as user name, exact device model or numbers, application may require outputs of very high similarity to their own ontology. When dealing with user information, privacy and security are high concern, especially with responsibility of conserving user data, in situation where credit card or passwords need to be given. Traditionally service and application providers are required to ensure security of their products and databases. In case of a data breach, the company who maintains and controls databases is held accountable. However, when using runtime discoverable widgets, developers cannot control what widgets are shown for each user yet those widgets are provided with sensitive user data if its output is deemed valid. This makes security liability unclear: which of widget provider, widget selector service or widget utilizer (i.e. service user is using) is held most accountable. Thus, nothing prevents malicious widget providers storing user information for their own purposes such as identity theft or selling them forward. Widgets that are determined as malicious can be removed from the discovery system and some kind of trust score could be given for each widget, but security should not be only reactive when issues arise. Users could be prompted to accept every used GUI component separately, giving a chance to determine if (s)he can accept requested information about him/her, but from usability standpoint it is not advisable to stop the user all the time. Therefore, discoverable semantic widgets are good tools for extending semantic and personalized user interfaces, but they do have big security issue to be solved. Another issue arises when widgets need to work together but input/output similarities are not high enough. Should developers nevertheless create one working fallback UI without personalization options in that case? It would diminish the benefit of faster development with automatic widgets. Otherwise, in best case scenario, user would have to view or fill data that is unnecessary for business logic, therefore making user experience worse even if it is somewhat personalized. In worst case scenario, business logic fails altogether when suitable widget is not found.

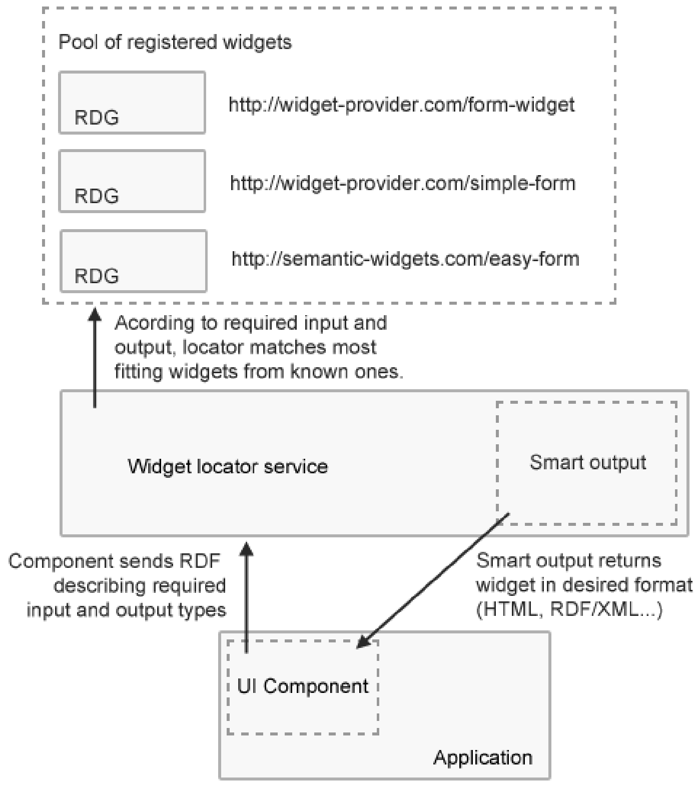

Pool of registered widgets
RDG
http://widget-provider.com/form-widget
RDG
http://widget-provider.com/simple-form
RDG
http://semantic-widgets.com/easy-form
Acording to required input and output, locator matches most fitting widgets from known ones.
Widget locator service
Smart output
Component sends RDF describing required input and output types
Smart output returns widget in desired format (HTML, RDF/XML...)
UI Component
Application

Figure 19. Widget locator service.

Above, some technical benefits and issues of semantic widgets were listed. Outside technical limitations, companies may have issues with UI consistency and therefore branding and brand recognition. Although usability of an application may become better for each individual user due to personalized GUI, application developers lose degree of control over consisted visual language of their software. Visual language is essential for user satisfaction as it will give more polished look and feel, which again can give competitive leverage over competitors, but may also be essential for a brand. Companies with well-known brands are strict on how they are represented. In commercial product development designers and developers need to be compliant to brand guidelines that define usage of colors, margins, images, look and feel etc. However, if GUI structuring and rendering is automatically generated at runtime, developers cannot guarantee that user experience is in line with brand rules. Additionally, while a widget may have user preferred logic and visual style, thus better usability, it may still conflict with other widgets in same application, thereby making overall user experience worse. In his paper Khiryenko used XHTML as an example output from remote visualization service. HTML is good for web content as it can be stylized with cascading style sheets (CSS), therefore helping to maintain visual consistency and brand guidelines. For semantic GUI personalization to work on other platforms, HTML output may not be suitable and different set of widgets is required. However, output type can be declared in widget module RDG and, in theory, it would be possible to make emulators or converters that modify widget output to target platform. This could be combined with widget locator service. Operating principle of centralized locator service is elaborated in Figure 19 below. Locator holds list of known widgets and developers can register URL of their widget to this locator service making them available for applications to use. Based on input and output description in widget's RDG, locator can then provide application most suitable widget using ontology matching. This can be an automated process. Locator can either hold widget descriptions, updating them time to time in case of changes or when widget becomes unavailable, or it can have widgets grouped (automatically or during widget registration process) so that it can always check their suitability at runtime. Some level of caching would be required to avoid excessive HTTP requests. Example of a single discoverable widget RDG is shown in Figure 20.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix sswap: <http://sswapmeet.sswap.info/sswap/> .
@prefix pd: <http://widget-provider.com/product-displayer/ontology.owl#> .

pd:productDisplayerWidget
   rdf:type sswap:Resource, pd:ProductDisplayerWidget ;
   sswap:providedBy pd:resourceProvider ;
   sswap:name "Product displayer" ;
   sswap:oneLineDescription "Widget returns product object with desired
visualization " ;
   sswap:operatesOn [
      rdf:type sswap:Graph ;
      sswap:hasMapping [
         rdf:type sswap:Subject, pd:ProductInfo ;
         pd:visualizationType [] ;
         pd:productName "";
         pd:productImageUrl "";
         […]
         sswap:mapsTo [
            rdf:type sswap:Object, pd:ProductObject;
         ] ;
      ] ;
   ] .
```

Figure 20. Example of a widget RDG.

Visual inconsistency is bad for user experience, but more easily manageable than inconsistencies with usage logic. That is, one widget behaves differently than others. Therefore, PSUP system should either strongly recommend best practices for widget developers, or even force widgets to be built on specific structure or framework. Discovery system can then favor widgets that follow these instructions. Also, as seen in Figure 14, every UI component can have standardized basic functionality for user initialized modifications in form of a recognizable cog-icon, thus making it easier for user to alter or change widget in a component slot. Common design pattern for widget modifications will give user the feel of consistency and control even if separate widgets have different internal behavior while giving developers freedom to build widgets as they see fit for achieving desired functionality. As developers are individuals with preferred coding styles, there will always be a level of inconsistency between applications in which suggested semantic widgets also belong.

Therefore, balance between required specifications (quality) and freedom (quantity) is needed.

While technical implementation (prototype) is not present, it is hard to determine if reusable widgets can work in practice. Security issues aside, lag can also cause usability issues as UI is created on-the-fly if discovery or server providing widget output is slow. Ideally there would be widgets to be found for each scenario, but in practice application developers are probably required to make fallback UI components for situations where fitting widget is not found, or one becomes unavailable. As a concept, however, semantic discoverable widgets can enhance personalization systems such as PSUP Framework.

3.7 Issues

As with other automated UI systems, one problem PSUP Framework also faces is unpredictability. Myers et al. puts it: "Automatic and model-based techniques have suffered from the problems of unpredictability. In fact, because heuristics are often involved, the connection between specification and result can be quite difficult to understand and control. Programmers must also learn a new language for specifying the models, which raises the threshold of use" (Myers, Hudson and Pausch 2000). In domain of user interface creation, that means developers cannot be certain how UI, especially if third party discoverable widgets are used, will look for the users. This will raise questions like:

- how do you make sure every required element is shown (e.g. in forms, no required input is hidden nor neglected)?
- how do you keep visual language consistent?
- is everything required shown and is the user aware of next step?

With automated interface systems, developer lose some control over the visuals and layouts, for example, division and order of the required inputs. Instead, they describe requirements on higher level and then trust for the system to provide logical enough user interface. In form views UI system may be providing user with input fields for required parameters and data, but they may be sorted illogically. That can cause confusion and worse conversion rate, which in commercial field is undesired. Illogical UI can produce unwarranted cognitive workload for user. Visual fidelity, aside from logical UI, is also part of user experience. If an application is working in very competitive market, developers must consider whether the best way to differentiate from competition is to focus on well-designed, handcrafted and polished UI or if user specific personalized UI is more important as they are contradicting concepts. PSUP Framework cannot guarantee both. Semantic vagueness and visualizing RDF data for unexperienced user is also hard. If user's profile is not large, it is likely that upon arrival to a website, (s)he does not have required information in his/her profile for matching against website's ontology. Therefore, user is opted to fill that information in terms defined by that website. At this stage, user probably does not need to see or understand RDF, but as the profile grows larger, need to view or edit profile data directly may arise. Visualizing all the user data in clear understandable way is a complex issue. Prefixes and domain names can take up big parts of screen space, especially on smaller mobile devices, making overall visualization cluttered (see Figure 15). Yet, those domain names are important for context. Therefore, smart data reductions, filters and search tools are required in PSUP management tools. As the profile grows, it will hold lot of information from which part of is likely duplicate entities. Storing data is not an issue in this day and clean up tools can be built to PSUP management tool (see chapter 3.4.3), but it is hard to predict what type of information profile will hold after years of building.

It is easy to falsely presume that personal, adaptive UI must have better usability. However, in practice system like PSUP requires lots of initial user input. As stated above, building user profile is a long endeavor, because it is not viable to make it complete in one step. Moreover, people change, preferences and abilities are not constant. Even physical abilities, such as eyesight deteriorate over time due to aging and that should be reflected in user's profile. One cannot predict all possible predicates and preferences to a profile, instead as it evolves over time, some contexts and domains become deprecated. This means that user cannot fill in information up front even if (s)he wants to. It would not only be tedious task, but simply impossible. Also, whenever websites update their ontologies, widgets may become unavailable, new ones are discovered and realignment is required, and user needs to confirm the new alignment. Realignments could be automated if matching systems evolve for accurate enough precision without user involvement. For example, using machine learning and services like WordNet. However, after profile is thorough enough realignment will become simpler task every time due to higher likelihood of existing information satisfying matching predicates. For this thesis, minor prototyping was done for PSUP maintenance application and ontology matching API by using Java Apache Jena. It gave indication of complexity of RDF visualization, but since no PSUP utilizing application project was conducted, actual end-user usability is hard to determine.

Possible issues outside technical implementation involve likelihood of traction PSUP Framework could get, and maintenance. Open sourcing codebase with suitable license is one way of building community and making sure PSUP system stays in active development even if small core team would be required for quality control and directing development in general. Semantic GUI system should not be considered as a single monolithic service on a server, but instead a network of independently maintained PSUP services (see end of chapter 3.5). With such open community project, there is an issue of who owns the profile data.

In Scientific American (2001) Tim Berner-Lee and co-authors predicted that open semantic web and agent technologies will revolutionize World Wide Web by bringing structure and meaningful content to Web pages. This semantic web is often referred as Web 3.0. Berner-Lee in his book Weaving the Web: "I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'semantic web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize." (Berners-Lee and Fischetti 2000). However, a survey from 2010 shows that his vision is yet to be realized. 52% of experts and 47% of participants of the survey agreed with the statement "By 2020, the semantic web envisioned by Tim Berners Lee will not be as fully effective as its creators hoped and average users will not have noticed much of a difference" (Anderson and Rainie 2010). Also, as mentioned in chapter 2.4.8, according to two-part survey in 2011, experienced developers and project managers listed lack of tools, skilled programmers, usability of available tools and standard ontologies as biggest barriers in applying Semantic Web Technologies. Biggest risks mentioned were exceeding time and cost estimations (Della Valle, Niro and Mancas 2011). Currently RDF and semantic web technologies are not much utilized as open data networks outside academic projects. For example, Google shut down its Freebase collaborative knowledge database in 2016, although, according to their announcement in 2014, it was to be replaced with their Knowledge Graph API (Google Inc. 2014). For PSUP Framework to succeed, or semantic GUI generation in general, semantic web technologies must gain more traction among software and web developers. However, it remains to be seen if companies will ever focus on utilizing versatile RDF technologies or whether it will be replaced by other means.

3.8 Future research

PSUP system can be refined in many ways. It could provide advanced features such as profile grouping: common properties from separate profiles can be mapped and used for forming groups of similar individuals. Comparing user to groups of people with similar habits and preferences, predictions of user behavior could be made in advance before user has had time to complete his/her profile. This would mean less user interference in form of filling up his/her information every time ontology match is not found, thereby giving better user satisfaction. For example, when a person, belonging to a group with similar properties in given domain goes to a new site, layout can be automatically adjusted based on average preference of similar group or multiple groups. This feature is one where advances in machine learning could be utilized. Machine learning is becoming more and more common tool in almost every area of life and web development should also benefit from it. Services like IBM Watson, Google Deep Mind and Microsoft Cognitive Services allow developers and researchers to offload resource heavy calculation to machines dedicated to it.

Privacy concerns have been mentioned multiple times in this thesis. This is largely because of the potential amount of data one person can have in one profile. Due to the nature of personalization, the more complete the profile is, the better PSUP system will work. However, this makes it a target for security attacks and therefore security must be priority for a system like this to work outside academic research and prototypes. One beneficial topic for a study would be finding ways to keep user data secure during alignments and possible automatic UI generation with third party widgets. Assuming PSUP Service is secure with proper measures, such as HTTPS requirement in all API calls, malicious intentions from a service utilizing PSUP, automatically discovered UI component (runtime UI generation) or man-in-the-middle between data transfers can pose threat for user data.

PSUP does not need to be limited to graphical user interfaces only. Semantic ontologies can define user preferences in general, therefore same profile can be used to adapt other interfaces, such as touch and voice recognition, as well. User can have preferences on how (s)he likes to hear things (language, accent, male/female) and interfaces with auditory feedback, such as GPS navigation tool, could adjust to user preference automatically. In future, smart home systems can also adapt to user's preferences and habits. Schema free semantic profiles could be used to transfer user data between smart home devices from different vendors. While semantic web technologies have not become as popular as Tim Berner-Lee predicted (see chapter 3.7), there have been multiple implementations and frameworks, such as Jena for Java, for developers to work with. On platform level, however, open semantic technologies are currently not available making OS level implementation harder, that is, full operating system adjusting to the same user profile as individual web applications making personal computers little bit more personal and blurring the line between application and operating system. When working on an operating system level, however, stable Internet connection cannot be guaranteed, and offline use of personal profile has to be available. This was discussed in chapter 3.2.

As this thesis is a theoretical description, working prototype would be beneficial for investigating how well semantic profile concept works in practice. Technologies mentioned in this thesis, such as RESTful architecture, ontology matching etc., are currently available and thus PSUP Framework could be prototyped. Usability problems, such as how often user action is required, would become more apparent in real use scenario. Should working prototype be built, open beta could be used for general feedback and ideas, and closed focus group test for investigating individual features of PSUP Framework. Also, designing easy-to-use (G)UI for management tool would be a task of its own as handling term context and different domains prefixes will easily make interface confusing for non-technologically oriented person.

4 Conclusion

This thesis described a framework for carrying personalized preferences from one application to another using combination of current semantic and non-semantic web technologies. While no working prototype is presented, many benefits for such system can be found. Main benefit of Personalized Semantic User Profile (PSUP) is that it stores user data in user's own words (semantic vocabulary), which then can be matched against application ontologies. Based on matched data between ontologies, applications can then adapt their graphical user interfaces or even business logic according to user's profile assuming developers have created variations from their application ontology. If not, developers could use third party widgets to build entire application UI or parts of it. This could save development resources and costs and give better user experience. User's PSUP is ever-adapting data store: as user spends time on different services, or modifies his/her profile directly via management tool, profile will evolve over time to reflect user's current identity and preferences. Semantic vocabularies are free from strict schemas: both users and developers are free to describe their profile and application in terms of their choosing. Ontology alignment is used to make sure these ontologies can be understood. Second benefit is that every service and application can choose how much semantic personalization is implemented: developer can create application ontology and only provide personalization options based on those terms. Thus, presentation can be assured to fit e.g. brand guidelines. On other spectrum, application could be built solely on independent UI widgets that output data objects that fit application logic.

Transferring user profile in semantic format can open lots of different possibilities, not only on higher conceptual level of individually tailored graphical user interfaces, but also on how we perceive applications and services in the future. As personalized applications behave and even look similar due to personal preferences, moving from one service to another can become virtually seamless, thus making learning curve smaller. Instead of remembering the use logic of every application, familiarity to ones used before will make learning of new applications easier. This would help especially older people who tend to be afraid of new GUIs. Therefore, services need to compete more on their business logic and other aspects, such as existing user base, instead of application visual fidelity alone. However, widely available personal profile has downsides. Main challenges for proposed PSUP Framework are currently privacy issues and adaptation rate of semantic web technologies. As PSUP can cover vast amount of user data, even sensitive information, it is likely a big target for malicious activity from phishing to identity theft. Framework described in this thesis does not cover all possible counter measures for such scenarios. Secure connections (SSL/TLS) are not enough if user data is given to third party widgets.

For PSUP Framework to succeed, it requires support from application developers to convince and make users to see benefits of creating their own profile to what many could view as "yet another thing that requires my information". That is, applications should offer real personalization options and consider them as differentiation from competition even if it requires more resources to build customization logic. However, considering the amount of commercial semantic web applications available today, knowledge and technical skill of current web developers is likely not very good for implementing semantic interfaces in their applications and services, and thus, decreasing the likelihood of easy adaptation of systems like PSUP Framework. This, as well as requirement for more development resources, can be addressed with independent semantic widgets that require less knowledge of underlying technologies from developers if appropriate widgets are available. UI widgets could be selected based on their output data so that they support given application's business logic. If user interfaces can be built, even partially, via autonomous, PSUP compatible widgets, adaptation would be faster. Although, as mentioned before, it opens issues with user privacy if data is given to third party. This was discussed in chapters 3.7 and 3.8. It remains to be seen if semantic web technologies will gain popularity among developers and companies or whether it will be replaced with other technologies.

References

Aas, Dag-Inge. *Authentication and authorization for native mobile applications using OAuth 2.0.* Norwegian University of Science and Technology, 2013.

Anderson, Janna Quitney, and Harrison Rainie. *The fate of the Semantic Web.* Pew Internet & American Life Project, 2010.

Benyon, David. *Accommodating individual differences through an adaptive user interface.* Vol. 10, in *Human Factors in Information Technology 10*, 149-149. 1993.

Benyon, David, and Dianne Murray. "Applying user modeling to human-computer interaction design." *Artificial Intelligence Review* (Springer Netherlands) 7, no. 3-4 (1993): 199-225.

Berners-Lee, Tim, and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor.* HarperInformation, 2000.

Bizer, Christian, Tom Heath, and Tim Berners-Lee. "Linked Data: The Story So Far." In *Semantic Services, Interoperability and Web Applications*, by Amit Sheth, 205-227. IGI Global, 2011.

Byron, Lee. *GraphQL issue tracker on Github.* 1 9, 2016. https://github.com/graphql/graphql-js/issues/271 (accessed 11 1, 2017).

David, Jérôme, Jérôme Euzenat, François Scharffe, and Cássia Trojahn dos Santos. "The Alignment API 4.0." *Semantic Web*, 2011: 3-10.

Della Valle, Emanuele, Gaetano Niro, and Catalina Mancas. "Results of a survey on improving the art of semantic web application development." *The 10th International Semantic Web Conference.* Bonn, Germany, 2011.

Dubray, Jean-Jacques. *Is OAuth 2.0 Bad for the Web?* 09 20, 2010. https://www.infoq.com/news/2010/09/oauth2-bad-for-web (accessed 11 27, 2016).

Eisenstein, Jacob, and Angel Puerta. *Adaptation in Automated User-Interface Design.* Palo Alto: ACM, 2000.

Euzenat, Jérôme. *Alignment API and Alignment Server.* September 05, 2016. http://alignapi.gforge.inria.fr/ (accessed January 08, 2017).

Euzenat, Jérôme, and Pavel Shvaiko. *Ontology Matching.* New York: Springer Berlin Heidelberg, 2007.

Feng, Xinyang, Jianjing Shen, and Ying Fan. "REST : An Alternative to RPC for Web Services." *2009 First International Conference on Future Information Networks.* Beijing: IEEE, 2009. 7 - 10.

Fernandez, Susel, Ivan Marsa-Maestre, Juan R. Velasco, and Bernardo Alarcos. "Ontology Alignment Architecture for Semantic Sensor Web Integration." *Sensors* 13, no. 9 (2013): 12581-12604.

Gajos, Krzysztof, Daniel Weld, and Wobbrock Jacob. "Automatically Generating Personalized User Interfaces with Supple." *Artificial Intelligence* 174, no. 12 (2010): 910-950.

Gessler, Damian DG, et al. *SSWAP: A Simple Semantic Web Architecture and Protocol for semantic web services.* 09 23, 2009. http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-10-309 (accessed 11 27, 2016).

Gessler, Damian. *Why SSWAP?* 07 26, 2014. https://sourceforge.net/p/sswap/wiki/protocol/ (accessed 11 28, 2016).

Google Inc. *Freebase announcement.* 12 16, 2014. https://plus.google.com/u/0/109936836907132434202/posts/bu3z2wVqcQc?cfem=1 (accessed 11 1, 2017).

Group, OAuth Working. *rfc6749 - the oauth 2.0 authorization framework.* 10 2012. http://www.rfc-base.org/rfc-6749.html (accessed 5 16, 2016).

Guarino, Nicola, and Pierdaniele Giaretta. "Ontologies and knowledge bases towards a terminological clarification." In *Towards very large knowledge bases: knowledge building & knowledge sharing*, 25-32. IOS Press, 1995.

Guarino, Nicola, Daniel Oberle, and Staab Steffen. "What Is an Ontology?" In *Handbook on Ontologies*, 1-17. Springer Berlin Heidelberg, 2009.

Hammer, Eran. *OAuth 2.0 and the Road to Hell.* 07 26, 2012. https://hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell/ (accessed 11 27, 2016).

Hartig, Olaf. *Querying a Web of Linked Data (Doctoral dissertion).* Humboldt-Universität zu Berlin, 2014.

Hayes, Philip J., Pedro A,. Szekely, and Richard A. Lerner. "Design alternatives for user interface management sytems based on experience with COUSIN." *CHI '85 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* New York, NY, USA: ACM, 1985. 169-175.

Hebler, John, Matthew Fisher, Ryan Blace, and Andrew Perez-Lopez. *Semantic Web Programming.* Indianapolis: Wiley Publishing, Inc., 2009.

Kadlec, Tomáš, and Ivan Jelínek. "Ontology-based Approach to Adaptation." *CompSysTech '07 Proceedings of the 2007 international conference on Computer systems and technologies.* ACM, 2007.

Kanjo, Daisuke, Yukiko Kawai, and Katsumi Tanaka. "How to build an adaptive web site: A Framework for User Adaptation." *WWW2004 Conference proceedings.* 2004. 252.

Khriyenko, Oleksiy. *Semantic UI: Automated Creation of Semantically Personalized User Interface.* GSTF Journal on Computing, 2015.

Lagun, Dmitry, Chih-Hung Hsieh, Dale Webster, and Vidhya Navalpakkam. "Towards Better Measurement of Attention and Satisfaction." *Proceedings of the 37th* *international ACM SIGIR conference on Research & development in information retrieval.* New York: ACM, 2014. 113-122.

Langley, Pat. "User Modeling in Adaptive Interfaces." *Proceedings of the Seventh International Conference on User Modeling.* Banff, Alberta: Springer, 1999. 357-370.

Lavie, Talia, and Joachim Meyer. "Benefits and costs of adaptive user interfaces." *International Journal of Human-Computer Studies* 68, no. 8 (August 2010): 508-524.

McCrae, John P., Francesca Quattri, Christina Unger, and Philipp, Cimiano. "Modelling the Semantics of Adjectives in the Ontology-Lexicon Interface." *Proceedings of the 25th International Conference on Computational Linguistics.* Dublin, Ireland, 2014. 198-209.

Mitchell, Tom. *Machine Learning.* McGraw-Hill Science, 1997.

Molich, Rolf, and Jacob Nielsen. *Improving a human-computer dialogue.* ACM, 1990.

Myers, Brad, Scott E. Hudson, and Randy Pausch. "Past, present, and future of user interface software tools." *ACM Transactions on Computer-Human Interaction (TOCHI)* 7, no. 1 (March 2000): 3-28.

Myllynen, Jenni. *Semanttinen web ja sukututkimus.* University of Jyväskylä, 2007.

Nielsen, Jacob. *Enhancing the explanatory power of usability heuristics.* Boston: ACM, 1994.

Otero-Cerdeira, Lorena, Francisco J. Rodríguez-Martínez, and Alma Gómez-Rodríguez. "Ontology matching: A literature review." *Expert Systems with Applications.* 2015. 949–971.

Puerta, Angel R, and David Maulsby. "Management of interface design knowledge with MOBI-D." *Proceedings of the 2nd international conference on Intelligent user interfaces.* Orlando: ACM, 1997. 249-252.

Santanen, Jukka-Pekka. ”Opinnäytteiden kirjoittaminen, lyhyt oppimäärä.” 2000. http://users.jyu.fi/~santanen/info/kirjoittamisesta.html (haettu 5.10.2012).

Structured Dynamics LLC. *Open Semantic Framework, Ontology Best Practices.* 1 12, 2014. http://wiki.opensemanticframework.org/index.php/Ontology_Best_Practices (accessed 4 19, 2016).

W3C. *Introduction to Model-Based User Interfaces.* 1 7, 2014. https://www.w3.org/TR/mbui-intro/ (accessed 11 5, 2016).

—. *Linked Data Standard.* Feb 26, 2015. https://www.w3.org/standards/techs/linkeddata (accessed Feb 22, 2016).

—. *OWL Web Ontology Language.* 2 10, 2004. https://www.w3.org/TR/owl-features/ (accessed 11 8, 2016).

—. *Resource Description Framework (RDF): Concepts and Abstract Syntax.* 02 10, 2004. https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (accessed 06 22, 2016).

—. *Semantic Web Standard.* 2015. http://www.w3.org/standards/semanticweb/ (accessed 4 12, 2016).

—. *SKOS Simple Knowledge Organization System.* 12 13, 2012. https://www.w3.org/2004/02/skos/ (accessed 08 03, 2016).

—. *SPARQL 1.1 Query Language Recommendation.* 3 21, 2013. https://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (accessed 11 8, 2016).

Vartiainen, Panu. *Using Metadata and Context Information in Sharing Personal Content of Mobile Users.* University of Helsinki, 2003.

Yampolskiy, Roman. "AI-Complete, AI-Hard, or AI-Easy-Classification of Problems in AI." *Proceedings of the Twenty-third Midwest Artificial Intelligence and Cognitive Science Conference.* Cincinnati, Ohio, 2012. 94-101.

Attachments

A Example of basic PSUP ontology

```
@prefix : <http://localhost:8080/PSUPService/ontologies/PSUPServiceOntology.owl#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://localhost:8080/PSUPService/ontologies/PSUPServiceOntology.owl> .

# This ontology contains main profile class and basic properties to start with

#################################################################
# Classes
#################################################################

# main root class
:Profile
   rdf:type
      owl:Class .

# helper class for grouping data
:SubProfile
   rdf:type
      owl:Class ;
   rdfs:subClassOf
      :Profile .

# helper class for grouping data
:Application
   rdf:type
      owl:Class .


# general item for ownership
:Item
   rdf:type
      owl:Class .
```

```
#####################################################################
# Basic data properties
#####################################################################

:name
   rdf:type
      owl:DatatypeProperty ;
   rdfs:domain :Profile;
   rdfs:range xsd:string .

   :firstname
      rdfs:subPropertyOf :name .


   :lastname
      rdfs:subPropertyOf :name .

:email
   rdf:type
      owl:DatatypeProperty ;
   rdfs:domain :Profile ;
   rdfs:range xsd:string .

:age
   rdf:type
      owl:DatatypeProperty ,
      owl:FunctionalProperty ;
   rdfs:domain :Profile ;
   rdfs:range xsd:int .

:birthdate
   rdf:type
      owl:DatatypeProperty ,
      owl:FunctionalProperty ;
   rdfs:domain :Profile ;
   rdfs:range xsd:dateTime .

:preference
   rdf:type
      owl:DatatypeProperty;
   rdfs:domain :Profile;
   rdfs:range xsd:string .

:restriction
   rdf:type
      owl:DatatypeProperty ;
   rdfs:domain :Profile ;
   rdfs:range xsd:string .

:handicap
   owl:equivalentProperty
      :restriction  .

:like
   rdf:type
      owl:DatatypeProperty;
   rdfs:domain :Profile;
   rdfs:range xsd:string .

:dislike
   rdf:type
      owl:DatatypeProperty;
   rdfs:domain :Profile;
   rdfs:range xsd:string .


######################################################################
# Object Properties
######################################################################

:hasSubProfile
   rdf:type
      owl:ObjectProperty ;
   rdfs:domain :Profile;
   rdfs:range :SubProfile .

   :hasRestrictions
      rdfs:subPropertyOf :hasSubProfile .

   :hasPreferences
      rdfs:subPropertyOf :hasSubProfile .

   :hasDislikes
      rdfs:subPropertyOf :hasSubProfile .

   :hasLikes
      rdfs:subPropertyOf :hasSubProfile .

# property for setting set of data for particular application
:forApplication
   rdf:type
      owl:ObjectProperty ;
```

```
    rdfs:domain :Profile;
    rdfs:range :Application .

# general object property for setting ownership of for example devices
:owns
    rdf:type
        owl:ObjectProperty;
    rdfs:domain :Profile;
    rdfs:range :Item .
```

B Example of user RDF

```
@prefix : <http://localhost:8080/PSUPService/id1/ontology#> .
@prefix psup: <http://localhost:8080/PSUPService/ontology/PSUPServiceOntology.owl#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://localhost:8080/PSUPService/id1/> .

:profile
    rdf:type psup:Profile ;
    psup:firstname "John" ;
    psup:lastname "Doe" ;
    psup:like "programming" ;
    psup:like "clouds" ;
    psup:age 29 ;

    psup:hasDislikes [
        rdf:type psup:SubProfile ;
        psup:dislike "overcast" ;
        psup:dislike "cold"
    ] ;

    # application specific data using application's ontology
    psup:hasSubProfile [
        rdf:type psup:SubProfile ;
        psup:forApplication <http://localhost:8080/ShopApplication> ;
        <http://localhost:8080/ShopApplication/id1/ontology#preferredCurrency>
<http://localhost:8080/ShopApplication/id1/ontology#euro>
    ] ;
```

```
# user's own term
:have [
   rdf:type :car ;
   psup:name "Toyota Yaris" ;
   :carMaker "Toyota" ;
   :carModel "Yaris" ;
]

.
```