Practical Use of Formal Concept Analysis in Service-Oriented Computing

Stéphanie Chollet, Vincent Lestideau, Yoann Maurel, Etienne Gandrille, Philippe Lalanda, Olivier Raynaud

Practical Use of Formal Concept Analysis in Service-Oriented Computing

Stéphanie Chollet[1], Vincent Lestideau[2], Yoann Maurel[2], Etienne Gandrille[2], Philippe Lalanda[2], Olivier Raynaud[3]

[1] Laboratoire de Conception et d'Intégration des Systèmes
F-26902, Valence cedex 9, France
stephanie.chollet@lcis.grenoble-inp.fr
[2] Laboratoire d'Informatique de Grenoble
F-38041, Grenoble cedex 9, France
{vincent.lestideau, yoann.maurel,
etienne.gandrille, philippe.lalanda}@imag.fr
[3] Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes
F-63173, Aubière cedex, France
raynaud@isima.fr

Abstract. Pervasive applications are encountered in a number of settings, including smart houses, intelligent buildings or connected plants. Service-Oriented Computing is today the technology of choice for implementing and exposing resources in such environments. The selection of appropriate services at the right moment in order to compose meaningful applications is however a real issue. In this paper, we propose a FCA-based solution to this problem. We have integrated FCA algorithms in our pervasive gateways and adapted them in order to allow efficient runtime selection of heterogeneous and dynamic services. This work has been applied to realistic use cases in the scope of a European project.

Keywords: Service-Oriented Computing, Pervasive environment, Service classification.

1 Introduction

Service-Oriented Computing (SOC) [13] brings software qualities of major importance. As with any planned reuse approach, it supports rapid, high quality development of software applications. Using existing, already tested, software elements is likely to reduce the time needed to build up an application and improve its overall quality. The key concept of the service-oriented approach is the notion of service. A service is a software entity that provides a set of functionalities described in a service description. The service description contains information on the service's functional part, but also on its non-functional aspects. Based on such specification, a service consumer can search for services that meet its requirements, select a compliant service and invoke it [6].

Web Services are the most popular and well-known technology for implementing Service-Oriented Architectures, both in the industry and in the academia.

A provider of Web Services can describe their service's functional and non-functional characteristics in a WSDL[4] file and then registers the service description in an UDDI[5] service registry. A client, or consumer, can search the UDDI registry for services that meet their requirements. Consumers use the SOAP[6] protocol to communicate with Web Services.

However, Web Services are not the only technology that implement the Service-Oriented approach. Web Services technology is dominant to integrate IT applications. However, in many other domains such as pervasive environments, the Service-Oriented approach is a solution of choice. Consequently, many technologies have been implemented and adapted to these domains. For instance, UPnP[7] or DPWS[8] are preferred in small area networks for devices whereas OSGi[9] and iPOJO[8] are often used in centralized and embedded equipments.

Service-Oriented Computing has thus evolved to support dynamic service discovery and lazy inter-service binding. Weak coupling between consumers and providers reduces dependencies among composition units, letting each element evolve separately. Late-binding and substitutability improve adaptability: a service chosen or replaced at runtime, based on its current availability and properties, is likely to better fulfill the consumer expectations. Such characteristics are essential when building pervasive applications with strong adaptability requirements. A key point is the ability to select at anytime the relevant service available in the registry to realize an application. The selection of services is a well-known complex problem. This problem has been particularly studied in the domain of Web Services [1, 3].

However, we think that this problem can not be restricted to a unique technology. Today, applications are composed of heterogeneous and dynamic services. Applications frequently need to integrate UPnP-based and DPWS-based filed devices and Web Services for remote applications. In this paper, we propose to tackle the problem of heterogeneous and dynamic service selection in the context of pervasive applications. In realistic use cases studied in the European OSAMI[10] project, one salient problem is the number of services to be considered. We investigate a solution based on Formal Concept Analysis (FCA) [10] to select pervasive services in a reactive and efficient fashion. Our proposition, that has been implemented and tested with the industrial partners of OSAMI, brings significant results in terms of efficiency and adaptability.

The paper is organized as follows: in Section 2, we detail the challenges of service selection in pervasive applications. In Section 3, some background about

[4] *Web Services Description Language*, http://www.w3.org/TR/wsdl.
[5] *Universal Description Discovery and Integration*, http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm.
[6] *Simple Object Access Protocol*, http://www.w3.org/TR/soap/.
[7] *Universal Plug and Play*, http://www.upnp.org.
[8] *Device Profile for Web Services*, http://specs.xmlsoap.org/ws/2006/02/devprof/devicesprofile.pdf.
[9] *Open Service Gateway initiative*, http://www.osgi.org/download/r4v41/r4.core.pdf.
[10] *Open Source Ambient Intelligence Commons for an Open and Sustainable Internet*, http://www.osami-commons.org/.

FCA is provided. Section 4 outlines the general idea of our FCA-based approach, detailed in Section 5: the service registry, the decision structure the selection algorithms. In Section 7, we present the implementation and experimental results. Before conclusion, Section 8 lists and discusses the related work.

2 Challenges of Service Selection

In Service-Oriented approach, a consumer must select the relevant service before invocation. The selection depends on the consumer requirements and service registry. The information shared between the different actors is the service description. For instance, from the consumer perspective, a Web Service is a black box that provides no technical details on its implementation. The only available information in the WSDL file includes the Web Service functionalities, certain characteristics such as non-functional properties, location and invocation instructions. Figure 1 recapitulates the different standards used by a set of service technologies for service description and service registry.

	Web Services	UPnP	DPWS	OSGi	iPOJO
Service Description	WSDL	UPnP Device Description	WSDL	Java Interface + Properties	Java Interface + Properties
Service Registry	UDDI registry	No registry	No registry	OSGi registry	OSGi registry
Service Discovery	Active	Active and passive	Active and passive	Active and passive	Active and passive
Service Notification and Service Withdrawal		Multicast	Multicast	Event	Event
Communication	SOAP	SOAP	SOAP	Java	Java

Fig. 1. Variety of standards implementing SOC.

Service discovery uses network protocols which allow automatic detection of services and/or devices. There are two kinds of service discovery: active - the consumer uses the service registry to discover the service - or passive - the service announces its arrival/departure on the network. Consequently, service technologies supporting passive discovery use multicast or event protocols. For instance, in pervasive environment, services are dynamic: smart devices join and leave the network at unpredictable times; back office applications are regularly updated. Services related to these devices are very volatile. In fact, devices connections and disconnections can be caused by many factors as diverse as users moves, battery problems, users demands, updates [11]. Then, two primitives (notification and withdrawal) can be added to support the dynamicity of services.

To conclude, there is a large variety of service technologies and they use heterogeneous technologies to describe, to discover and to communicate. Note

that the service selection is complex due to this multitude of technologies and to the dynamicity of services.

In addition, today applications are more and more composed of multiple heterogeneous services. For exemple, an application for acquisition chain requires devices (UPnP and/or DPWS) to acquire data and services for the business (Web Services) such as to analyze or to store data. We have extended the basic SOC pattern in order to support heterogeneity (Figure 2).

A key issue in such context lies in the runtime selection of relevant services in environments filled with devices and applications. Service selection has become a challenge in pervasive environments with the increasing number of dynamic devices, often providing close functionalities but with different technologies and different descriptions.

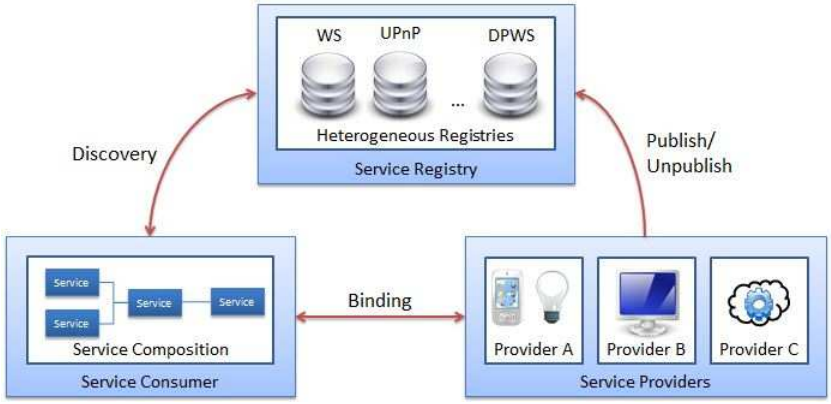

Fig. 2. SOC adapted to the service heterogeneity.

3 Theoretical Fundations: Formal Concept Analysis

Formal Concept Analysis (FCA) [10] is a theoretical and mathematical framework used to classify. We propose to use the Formal Concept Analysis method to classify available services at runtime. We very shortly define the main concepts of FCA. The purpose of FCA is to build a partially ordered structure, called concept lattice, from a formal context.

Definition 1. *A **formal context** $\mathbb{K}$ is a set of relations between objects and attributes. It is denoted by $\mathbb{K} = (O, A, R)$ where O and A are respectively sets of **Objects** and **Attributes**, and R is a **Relation** between O and A.*

Definition 2. *A **formal concept** C is a pair (E, I) where E is a set of objects called **Extent**, I is a set of attributes called **Intent**, and all the objects in E are in relation R with all the attributes in I.*

Thus, the Extent of a concept is the set of **all** objects sharing a set of common attributes, and the Intent is the set of **all** attributes shared by the objects of the Extent. Formally:

- $E = \{o \in O, \forall i \in I, (o, i) \in R\}$,
- $I = \{a \in A, \forall e \in E, (e, a) \in R\}$.

Consequently, a formal concept $C = (E, I)$ is made of the objects in E which are exactly the set of objects sharing the attributes in I.

Let X a set of attributes. We define the function $Closure_{\mathbb{K}}(X)$ which associates to X the concept made of the set of objects sharing X and the other attributes shared by this set of objects. Note that the computation of a formal concept from a set of attributes X of size n has a complexity of $\mathcal{O}(n \times m)$ where m is the number of objects.

The set $\mathcal{C}(\mathbb{K})$ of all concepts induced by a context can be ordered using the following partial order relation: $(E_1, I_1) <_{\mathcal{C}} (E_2, I_2)$ if $E_2 \subset E_1$ and $I_1 \subset I_2$.

Definition 3. *A **concept lattice** is defined as the pair $(\mathcal{C}(\mathbb{K}), \leq_{\mathcal{C}})$. Let two concepts (E_1, I_1) and (E_2, I_2) we say that (E_2, I_2) is a successor of (E_1, I_1) if $(E_1, I_1) <_{\mathcal{C}} (E_2, I_2)$. Given I_1 a subset of A, we note by **successors**(I_1) the set of successors of the concept (E_1, I_1). The concept lattice can be represented by a particular graph called Hasse Diagram.*

Note that the computation of a concept lattice from a formal context has a complexity of $\mathcal{O}((n+m) \times m \times |\mathcal{C}(\mathbb{K})|)$ where n is the number of attributes and m is the number of objects ([12]). Most of the time we have $n << m$ and the complexity becomes $\mathcal{O}(m^2 \times |\mathcal{C}(\mathbb{K})|)$.

4 Global Approach

It is assumed that we have an application which is a composition of abstract services defined at design time. During this step, a set of architectural constraints are defined. These constraints are the expected functionnalities and/or non-functional properties; they are considered as mandatory features. From design time to runtime we must consider as much on these architectural constraints as runtime environment constraints (*i.e.* arrival and departure of services).

It is required to select a concrete service for each abstract service of the composition. Current approaches only select services from architectural constraints. In addition, these approaches realize service selection just before runtime. Consequently, they do not completely handle the environment dynamicity.

Our objective is to ensure at runtime the more adapted configuration according to the architectural constraints, the current environment and user preferences. Figure 3 illustrates our approach which is divided into three parts:

- **Setting forth user requirements.** The user requirements are expressed by a request which contains the architectural constraints. These constraints are composed of mandatory and optional features defined by the architect at design time.

– **Storage of available services in the environment.** The service registry supports the dynamicity and the heterogeneity of services. It is global and it contains the service descriptions annotated with user properties (QoS).
– **Service selection.** The selection process is divided into two parts. First, a set of services is selected and classified according to the defined user requirements. A decision structure is built at that time. Second, one approriate service must be chosen from this structure. We define adapted algorithms to search in this structure the most appropriate service according to the user preferences.

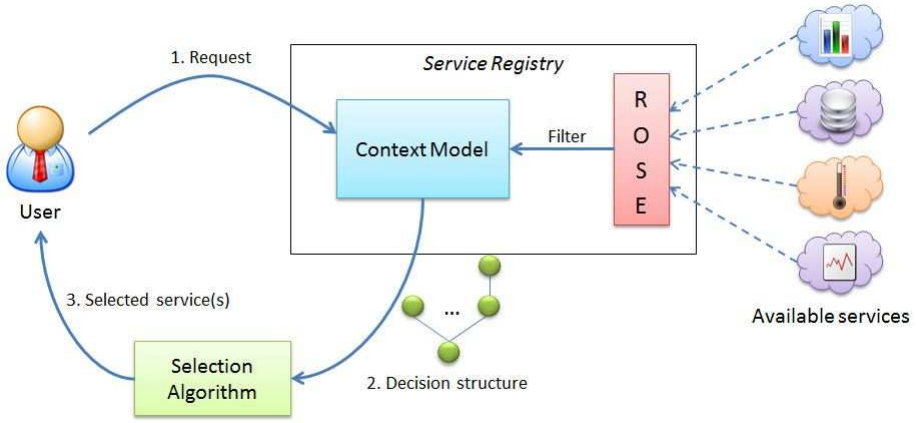

Fig. 3. Global approach.

The following section is divided into three parts in which we detail the service registry, the computation of the decision structure and the selection algorithms.

5 Application to Services

5.1 Service Registry

The service registry is the central element of our approach. Its goal is to maintain a global view of all the available services at runtime. It is composed of two parts:

– ROSE [4]: an integration platform monitoring the runtime environment, *i.e.* it traces services availability and provides information about them;
– a context model: it stores and maintains the information recovered by ROSE.

ROSE is an OSGi-based open source middleware[11]. It detects all the services being in the environment. This tool is able to support active and passive service

[11] http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/Rose

discovery. Concretly, it supports multicast and event mechanisms used to service notification and service withdrawal. These capabilities are essential because services related to devices are very volatile.

Since ROSE detects all the services being in the environment, we have defined a filter to compute an application-specific context model. The filter allows to specify the services of interest. In fact, only the services of interest for the application are selected. For instance, a multimedia entertainment application requires multimedia services such as movies library, TV... Humidity sensors will be ignored.

In our project, we have adapted the context model to the FCA formal context. The context model can be seen as a relation between the filtered services and the possible service features of the application domain. We can categorize the service features into three main groups (Table 1):

- The service technologies (Web Service, UPnP, DPWS...),
- The service functionalities,
- The non-functional properties required and/or provided by the service.

	$t_1, ..., t_i$	$f_1, ..., f_j$	$nf_1, ..., nf_k$
s_1			
...			
s_n			

Table 1. Context model as a formal context.

We have tested our work with security properties (authentication, confidentiality, integrity...), a particular non-functional property that can be expressed as a boolean value (*i.e.* services are secured or not). However, all non-functional properties are not boolean. But, in [15, 9, 2], authors have investigated the link between FCA with numerical data.

5.2 Decision Structure

From the context model expressed as a formal context, we extract a set of ordered formal concepts, *i.e.* an extract of the concept lattice. Given that the computation of a lattice has a complexity in $\mathcal{O}(m^2 \times |\mathcal{C}(\mathbb{K})|)$ and that the space complexity is in $\mathcal{O}(2^n)$ since the number of concepts is potentially 2^n, it is not realistic to dynamically compute the entire concept lattice for each user request. We propose a way to compute only the interesting concepts.

In the concept lattice, we can distinguish two exclusive groups of concepts, as illustrated in Figure 4:

- **concepts with no real meaning.** These concepts contain in their intent a set of properties which is not usable. For example, all the concepts with an intent composed of only non-functional properties do not make sense. The

top and the *bottom* of the lattice are also meaningless. The *top* contains in its intent all the attributes, *i.e.* all the functional and non-functional properties, and the extent is empty because no service can provide all the properties. Similarly, the *bottom* contains in its extent all the services and the intent is empty because it is not possible to have a common property for all the services. For example, the type of service is an exclusive property.

– **concepts with sense.** Contrary to the previous group, the intent of the concepts makes sense, *i.e.* the intent contains coherent information. For example, at least one functionality is in the intent.

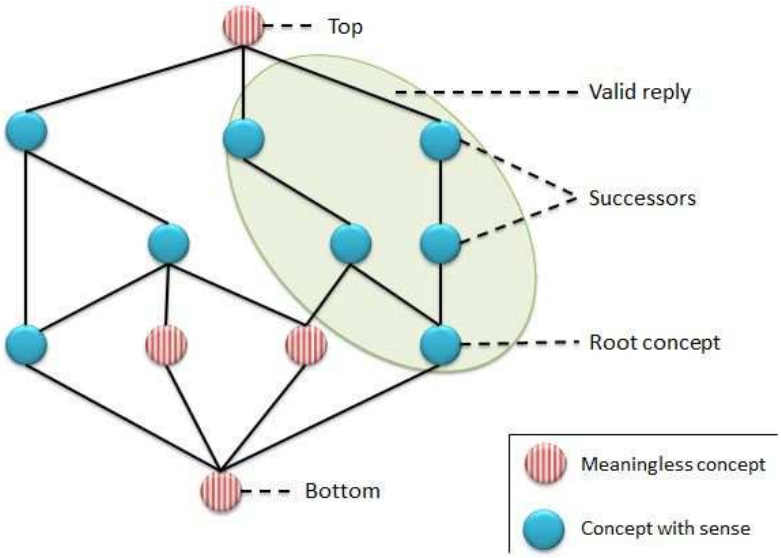

Fig. 4. Computation of the decision structure.

This classification into concepts with or without applicative meaning is the key of our approach. According to concept semantics, we can compute only the interesting concepts and not the entire lattice. The interesting concepts are a subset of meaningful concepts extracted from the lattice. The subset is a structure where the root element is a formal concept and the nodes are the successors of the formal concept.

The decision structure is computed in response of a user request. In the following, a user request (denoted in bold) is defined by a set of mandatory features (denoted by MF). The result of the request is set of formal concepts in which the extent (denoted by S) contains all services sharing a set of common features, the mandatory features and possibly a new set of found features (denoted by FF).

In the following, we present two kinds of user requests based on the selection of a service for a workflow activity. First, the selection can be only based on mandatory features. Second, the selection can be based on mandatory and optional features.

The solution for the selection of services from a set of mandatory features is the computation a formal concept called root concept in which the intent contains the mandatory features: $(S; \mathbf{MF} \cup FF)$. The sets S and FF can be empty. If the extent S is empty, there is no service available providing the mandatory features.

The selection based on mandatory and optional features is an extension of the previous selection. To take into account the optional features, we propose to compute the successors of the root concept $(S; \mathbf{MF} \cup FF)$. The computation of the succesors is then an extract of the concept lattice that can be viewed as the decision structure: $(S; \mathbf{MF} \cup FF) \cup \texttt{successors}(\mathbf{MF} \cup FF)$. This decision structure is limited to the set of successors in the concept lattice.

Table 2 is an illustration of a context model. The context model is a simplification of a real context model because we have only few service characteristics (attributes) and eight available services. For clarification purposes, the extract of context model only contains the functional attributes *Temperature* and *Humidity* and only the services providing the *Temperature* functionality.

	WS	UPnP	DPWS	Temperature (T)	Humidity (H)	Authentication (A)	Confidentiality (C)	Integrity (I)
S_1		X		X				X
S_2		X		X			X	X
S_3			X	X	X			X
S_4			X		X		X	X
S_5		X		X	X			X
S_6		X		X		X		
S_7		X		X				
S_8			X	X				X

Table 2. Extract of the context model.

For example, the administrator can select all the services providing the temperature functionality. The selection result for *Temperature* activity is the formal concept $(\{S_1, S_2, S_3, S_5, S_6, S_7, S_8\}; \{\mathbf{Temperature}\})$. All the services except S_4 provide the *Temperature* functionality. The administrator can refine his request: the classification of the *Temperature* services can be computed from the concept $(\{S_1, S_2, S_3, S_5, S_6, S_7, S_8\}; \{\mathbf{Temperature}\})$ previously obtained. The successors of this concept constitutes the decision structure (Figure 5).

At the bottom of the figure, we find the concept $(\{S_1, S_2, S_3, S_5, S_6, S_7, S_8\}; \{\mathbf{Temperature}\})$. Services are classified according to their characteristics. In [7],

we have extended our approach to the composition of decision structures in response to complex user requests.

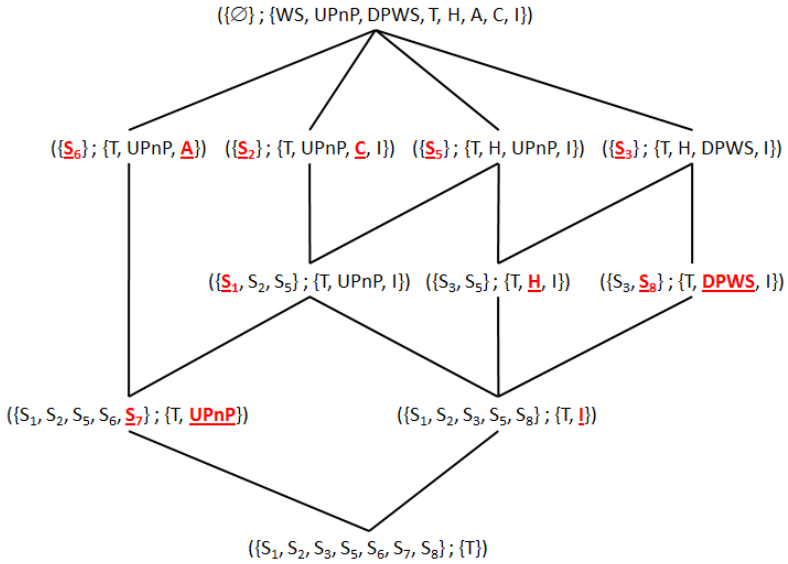

Fig. 5. Example of an extract lattice for *Temperature* activity.

5.3 Algorithms for Selection

Depending on the expected response of the request (one or more services), it may be necessary to choose among all possible services. For this, it is possible to use different selection algorithms:

A selection based only on mandatory features: computation of a formal concept $(S; \mathbf{MF} \cup FF)$**.** Some naive but efficient algorithms with a complexity $(\mathcal{O}(1))$ will return one service among the extent S (*e.g.* "first", random). If the selected service is not available at runtime (realistic use case in pervasive computing), it is possible to search the extent S to choose another available service (algorithms with a complexity $(\mathcal{O}(n))$. The ability to use the extent without recomputing the formal concept is a major advantage of our solution.

A selection based on mandatory and optional features: computation of a decision structure $(S; \mathbf{MF} \cup FF) \cup \texttt{successors}(\mathbf{MF} \cup FF)$**.** In a decision structure, services are classified according to a set of optional features defined by the user at specification time. Being guided by selection policies (*i.e.* QoS-based, target environment description-based) when exploring the decision structure allows fine-grained service selection. In this case, the time complexity of the selection is linear in the number of successors of the root concept in the concept lattice.

For instance, with the decision structure previously built in Figure 5, an optional criterion for the user request *Temperature* can be the services implemented in the UPnP technology. Then, the right side of the tree can be pruned. Services S_1, S_2, S_5, S_6 and S_7 provide the functionality *Temperature* with an UPnP implementation.

In another example, let us consider that the user wants at least two *Temperature* services with, if possible, confidentiality and integrity properties for the data exchange. Only service S_2 provides the confidentiality (C) and integrity (I) properties and it is also implemented with UPnP technology ($\{S_2\}; \{T, UPnP, C, I\}$). However, thanks to the decision tree, the user can relax the constraints. Services S_1 and S_5 have the same features than S_2 but the confidentiality property ($\{S_1, S_2, S_5\}; \{T, UPnP, I\}$).

The ability to use the decision structure at runtime (without rebuilding) is also a major advantage of our solution.

6 Reacting to Service Availability at Runtime

In our approach, the decision structure is stored in memory in order to be reactive to events related to services (departure and re-arrival). In addition, we keep a pointer to the formal concept from which the service was selected. For instance, the service S_1 is a selected service extracted from the formal concept ($\{S_1, S_2, S_5\}; \{T, UPnP, I\}$) (Figure 5).

At runtime several scenarios can occur. For example, in the case of service departure, several options are possible:

- First, we can choose a replacement among the other services of the extent of the pointed concept (subject to the service availability). In the example, we can consider that S_1, S_2 and S_3 are owned by a same equivalent class; they have the same features.
- Second, if there is no available service in the extent of the pointed concept, we can use the decision tree for backtracking at runtime. More specifically by selecting a service among the extents in one (or more) of the predecessors (*i.e.* $\{S_1, S_2, S_5, S_6, S_7\}$ and ($\{S_1, S_2, S_3, S_5, S_8\}$). It is also possible to restart a partial tree search by relaxing a few constraints.

In this way, we can quickly and easily (at runtime) find a replacement service while ensuring the most appropriate configuration. In addition, our approach can also take into account the re-arrival of a more appropriate service.

7 Implementation and Validation

In this section, we present the implementation of our approach and the experimental results. This work has been validated in the European OSAMI project.

7.1 Implementation

To validate our approach, we made a Java implementation. Our implementation has been cut into three modules (Figure 6):

- A data acquisition module;
- A processing module, containing an algorithm for the formal concept computation;
- A renderer module, to display the results.

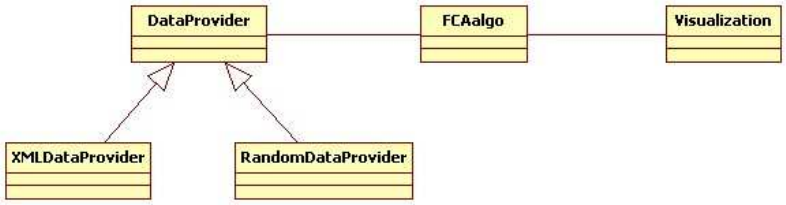

Fig. 6. Application architecture.

For the needs of our experiment, two implementations have been realized. The first one loads static data from an XML file; the second uses a random number generator to generate data. Loading data from a static XML file allowed us to test our algorithm on a test bench; using the random number generator, we were able to test the robustness of our algorithm for big data sets. Data generation is performed in several stages. The first one generates the communication protocol (UPnP, DPWS), and the following ones the associated functional and non-functional properties.

To obtain valid data using random generation, this one was made in several stages. In every stage, attributes laws of generation are function of the the previous stages results. Then specificities associated with every technology are taken into account.

In the processing module, to obtain short processing times and low memory footprint with big data sets, we used bit fields, and took care of allocating only the strictly necessary memory.

The last part which is the renderer module, is implanted using JGraph[12], a graph drawing open source software component written in Java. This graphic feedback allowed a fast interpretation of the obtained lattices.

7.2 Experimental Results

In this section, we present the result of our experimentations. The goal of this work is to prove the feasability of our approach, *i.e.* introducing FCA in the

[12] http://www.jgraph.com/

service selection. Then, we have evaluated our work with a performance study. Experimentations have been made on a 2.20 GHz Intel Core Duo, 4GB of memory, Windows 7, 32 bits. We have fixed to 24 the number of attributes for the context model (3 technologies, 11 functionalities and 10 security properties).

Number of computed concepts To test the feasability of our approach, we have evaluated the number of computed concepts according to the number of available services and the size of the request. The request contains:

- No constraint, *i.e.* equivalent to compute the entire lattice,
- One functional constraint, *i.e.* the minimal use case because the user knows at least the expected functionality,
- One functional constraint and the technology used to implement the service.

For this experimentation, we count the number of computed concepts (Figure 7) for these requests according to the available services in the context model.

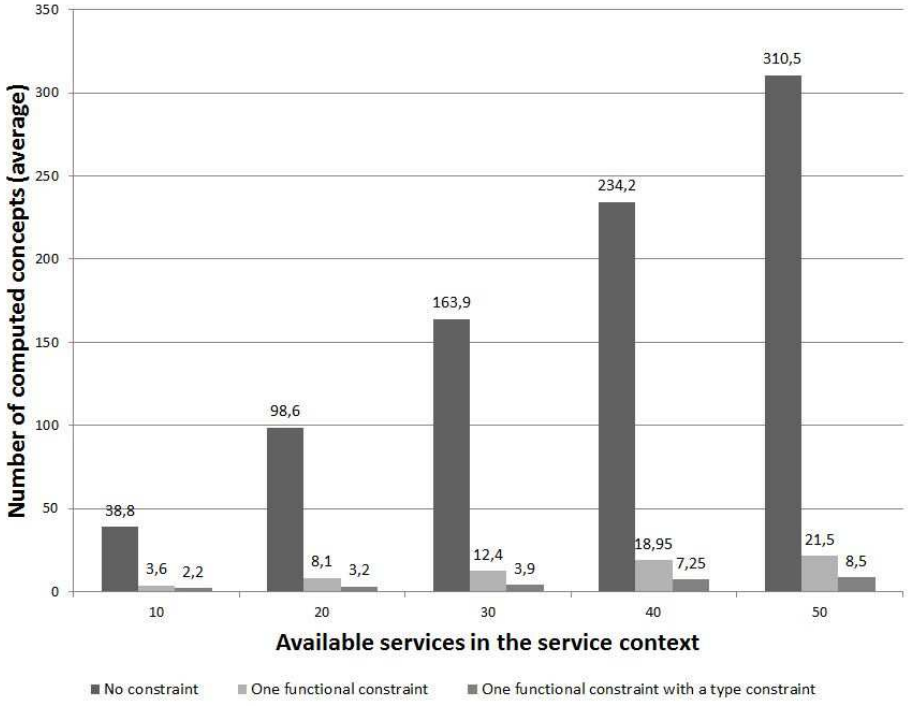

Fig. 7. Number of computed concepts in function of the available services.

We note that the computation of only interesting concepts largely decreases the number of computed concepts. For a request based on one functionality,

the decrease is 92% in average; for the second type of request, the decrease is 96%. Consequently, we have studied the computation time for evaluating the performance of our approach.

Computation time The major inconvenience of using FCA is the complexity of algorithm to compute a lattice: $\mathcal{O}((n+m)\times m\times|\mathcal{C}(\mathbb{K})|)$ where n is the number of attributes (properties) and m is the number of objects (available services). Even if we have decreased the number of computed concepts, we compute an ordered set of concepts. In a pervasive environment, we must propose this set in a "reasonable" time, *i.e.* due to the dynamicity of the application, and support a registry containing numerous services. For instance, a large building or plant can be approximately composed of one thousand devices (services). To test the reactivity of our approach, we have studied the computation time.

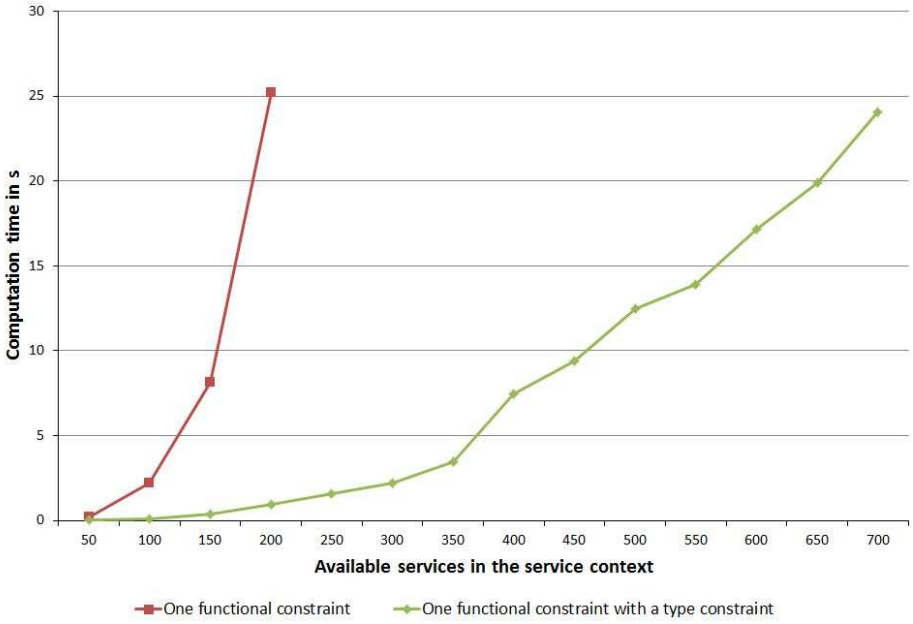

Fig. 8. Computation time as a function of the available services.

In Figure 8, we have represented the computation time for two types of requests. We have not represented the computation time for a request with no constraint but it is approximately 160s for 50 available services and 2700s for 100 available services. This is clearly not realistic in a pervasive environment. However, for the selection, the request contains at least the service functionality and the computation time stays "reasonable" (25s) with 200 available services

in the context. The request with one functionality constrained by a technology - more realistic request - can be executed also in 25s but for 700 available services.

8 Related Work

The problem of service selection, depending on service classification and FCA, has been studied by a few authors. Bruno et al. [5] propose an approach based on machine-learning techniques to support service classification and annotation. Peng et al. [14] classify Web Services in a concept lattice. Services are classified according to their functional operations regardless of non-functional aspects. Azmeh et al. [3] classify Web Services by their calculated QoS levels and composability modes. This classification is made with a Relational Concept Analysis approach, an extension of FCA. In these approaches, the inconvenient of using FCA is the large size of the concept lattice even if optimizations such as minimizing the number of attributes are proposed. However, these approaches are interesting in the assumption that the concept lattice is computed only once. But in the pervasive domain, services regularly appear and disappear, which means recalculating the lattice. Moreover, these approaches can not manage simultaneously different technologies (UPnP, DPWS...).

The key element of our approach is the service registry definition. We have adapted the service registry to the FCA context model. Ait Ameur [1] proposes to adapt the registry to a semantic registry in which semantic Web Services are stored. The introduction of ontologies allows to define a subsumption relationship between services that expresses a substituability relationship between these services. In our approach, ontologies can be added in the filter of the service registry in order to minimize the number of attributes in the context model.

9 Conclusion

Pervasive applications are made of heterogenous and dynamic services requiring a runtime adaptability and context-sensitive selections. In this paper, we have proposed an FCA-based solution for this selection problem. The integration and adaptation of FCA in our pervasive platform allows efficient runtime selection of heterogeneous and dynamic services. The characteristics of formal concept and decision structure avoid reiterate each time the selection algorithm which significantly improves performance at runtime. Our work has been applied on realistic use cases of the OSAMI European project and the results of experimentation show that it is possible to integrate FCA-based approach in dynamic context.

In the future, we will be interested in automating the selection in the decision structure of the appropiated service. We propose to replace the user by an autonomic manager to be more reactive to the context.

References

1. Y. Ait-Ameur. A semantic repository for adaptive services. In *IEEE Congress on Services*, pages 211–218, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

2. Z. Assaghir, M. Kaytoue, W. Meira, and J. Villerd. Extracting decision trees from interval pattern concept lattices. In *Concept Lattices and their Applications*, 2011.
3. Z. Azmeh, M. Driss, F. Hamoui, M. Huchard, N. Moha, and C. Tibermacine. Selection of composable web services driven by user requirements. In *ICWS 2011, IEEE International Conference on Web Services*, pages 395–402, Los Alamitos, CA, USA, 2011. IEEE Computer Society.
4. J. Bardin, P. Lalanda, and C. Escoffier. Towards an Automatic Integration of Heterogeneous Services and Devices. In *Proceedings of IEEE Asia-Pacific Services Computing Conference*, pages 171–178, Los Alamitos, CA, USA, 2010. IEEE Computer Society.
5. M. Bruno, G. Canfora, M. D. Penta, and R. Scognamiglio. An approach to support web service classification and annotation. In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, pages 138–143, Los Alamitos, CA, USA, 2005. IEEE Computer Society.
6. S. Chollet, P. Lalanda, and J. Bardin. Service-Oriented Computing: from Web Services to Service-Oriented Components. In I. Global, editor, *Service Life Cycle Tools and Technologies: Methods, Trends and Advances*, pages 1–20. Jonathan Lee and Shang-pin Ma and Alan Liu, November 2011.
7. S. Chollet, V. Lestideau, P. Lalanda, Y. Maurel, P. Colomb, and O. Raynaud. Building FCA-based Decision Trees for the Selection of Heterogeneous Services. In *SCC '11: Proceedings of the 2011 IEEE International Conference on Services Computing*, pages 616–623, Washington, DC, USA, July 2011. IEEE Computer Society.
8. C. Escoffier, R. S. Hall, and P. Lalanda. iPOJO: an Extensible Service-Oriented Component Framework. In *IEEE International Conference on Services Computing (SCC)*, pages 474–481, Los Alamitos, CA, USA, 2007. IEEE Computer Society.
9. B. Ganter and S. O. Kuznetsov. Pattern structures and their projections. In H. S. Delugach and G. Stumme, editors, *9th International Conference on Conceptual Structures (ICCS 2001)*, volume 2120 of *Lecture Notes in Computer Science*, pages 129–142. Springer, 2001.
10. B. Ganter and R. Wille. *Formal Concept Analysis - Mathematical Foundations*. Springer, Berlin, Heidelberg, 1999.
11. P. Lalanda, J. Bourcier, J. Bardin, and S. Chollet. Development of service-oriented pervasive home applications. In I. Book, editor, *Smart Home Systems*, pages 1–16. Mahmoud A. Al-Qutayri, February 2010.
12. L. Nourine and O. Raynaud. A fast incremental algorithm for building lattices. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2-3):217–227, 2002.
13. M. P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *Proceedings of the fourth International Conference on Web Information Systems Engineering*, pages 3–12, Los Alamitos, CA, USA, December 2003.
14. D. Peng, S. Huang, X. Wang, and A. Zhou. Management and retrieval of web services based on formal concept analysis. In *Proceedings of the The Fifth International Conference on Computer and Information Technology*, pages 269–275, Los Alamitos, CA, USA, 2005. IEEE Computer Society.
15. G. Polaillon. Interpretation and reduction of galois lattices of complex data. In Springer-Verlag, editor, *Advances in Data Science and Classification*, pages 433–440. A. Rizzi and M. Vichi and H.-H Bock, 1998.