

# Building FCA-based Decision Trees for the Selection of Heterogeneous Services

Stéphanie Chollet, Vincent Lestideau,  
Philippe Lalanda, Yoann Maurel  
*Laboratoire d'Informatique de Grenoble*  
F-38041 Grenoble cedex 9, France  
{stephanie.chollet, vincent.lestideau,  
philippe.lalanda, yoann.maurel}@imag.fr

Pierre Colomb, Olivier Raynaud  
*Laboratoire d'Informatique, de Modélisation  
et d'Optimisation des Systèmes*  
F-63173 Aubière cedex, France  
{pierre.colomb, olivier.raynaud}@univ-bpclermont.fr

**Abstract**—Late-binding and substitutability offered by the service-oriented approach improve adaptability but increase the need for fast and efficient algorithms to select services. In this paper, we proposed to use the Formal Concept Analysis (FCA) approach as a classification tool to select services at run time, according to user specifications. We propose to classify existing services and generate a decision tree to help user select the most appropriate service(s). One of advantages of using FCA is the ability to select without additional cost an equivalent service in the case of a service must be replaced at runtime. Our approach have been implemented and validated on pervasive use cases within a European collaborative project.

**Keywords**-Service selection, Service classification, Functional and non-functional properties, Formal Concept Analysis.

## I. INTRODUCTION

The emergence of service-oriented computing has facilitated the development and deployment of pervasive applications, providing assistance to people in their living environments. Most applications and devices are today exposed as services and can be used in accordance with the service pattern. In the plant floor, for instance, more and more UPnP or DPWS devices are manufactured by major actors in automation and control. Even embedded applications, used in building or plants, are new exposed as Web services.

Obviously, service orientation comes with software qualities of major importance. As with any planned reuse approach, it supports rapid, high quality development of software applications. 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. That being said, it is complex to conceive and implement an application made of dynamic, heterogeneous services and required to meet non functional requirements like security.

A key issue in such context lies in the runtime selection of relevant services in environments stuffed with devices and application. Services selection has become a challenge because of the increase of the number of devices, often providing close functionalities but with different technologies and

different descriptions. No surprisingly, there is no common format for description and no shared registry capabilities. To solve this problem of heterogeneous descriptions, some approaches have focused on classification of (web) services. Azmeh *et al.* propose a tool named WSPAB [1] that aims to define a complete solution for facilitating the task of finding the most relevant Web Service. They use formal concept analysis approach to classify Web Services and it is based on the assumption that two operations are equivalent if they have the same signature. However this criterion is not always relevant because service functionalities cannot always be described explicitly by signatures. Bianchini *et al.* [2] provides ontology to organize services in the form of e-services and to improve the service discovery.

Regarding selection many algorithms have been proposed to select services trying to find the best service. Since the concept of best service is rather subjective, most of these works have introduced non-functional characteristics and more particularly Quality of Service (QoS) criteria such as response time, throughput, availability and reliability. These quality-driven algorithms ensure that the selected services meet the functionalities and the QoS requirements. Some of these algorithms are based on brute-force approach of seeking all possible solutions [3]. However these solutions have a high cost in time and resources, which is a drawback in a pervasive environment. To reduce the search time other approaches have proposed heuristic-based solutions. Mabrouk *et al.* [4] present an algorithm taking into account the concept of dynamic binding allowing composition with on-the fly services. Canfora *et al.* [5] propose to extend QoS-based solutions to take into account the functional and non-functional characteristics. These heuristic-based algorithms make the selection more suitable for pervasive environments but they focus mainly on web service technology and non-functional criteria related to QoS.

Composing dynamic, heterogeneous services is however not an academic fantasy! Applications frequently need to integrate UPnP-based and DPWS-based field devices and Web Services for remote applications. Most services are dynamic: smart devices join and leave the network at unpredictable times; back office applications are regularly updated. In

addition, security has to be considered when building a service-oriented application. In this paper, we will build on a use case developed with Thales inc. in the European SODA<sup>1</sup> project. It describes an alarm management system. The system collects physical measures from the real environment like temperature and humidity. Data are gathered, analyzed, and then recorded. Finally, based on the analysis, actions (storage, notification, and action on a machine) can be triggered by the system. This alarm management system can be implemented as a service orchestration as it is illustrated by Figure 1.

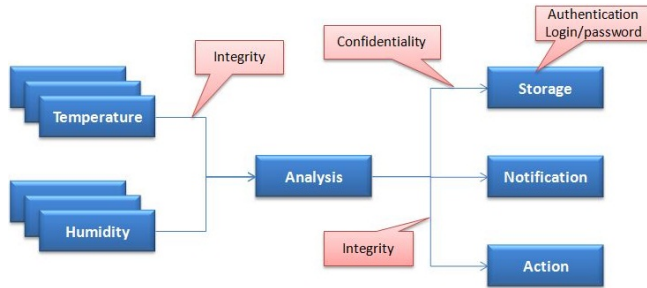


Figure 1. Specification of an alarm system example.

The paper is organized as follows. First, theoretical foundations of the Formal Concept Analysis (FCA) are given. Section III deals with our FCA-based approach to select appropriate services. Section IV details the different kinds of classification and selection according to user request. Section V presents the implementation of our approach. Before the conclusion, we discuss the advantages of our work.

## II. THEORETICAL FOUNDATIONS

We propose to use the Formal Concept Analysis method to classify services at runtime as a function of user specification. The goal of this classification is to provide flexibility and efficiency in service selection.

Formal Concept Analysis (FCA) [6] is a mathematical classification tool. It is used in many practical cases in many domains including software engineering [7], data-mining [8], and linguistics [9]... The purpose of this method is to build a partially ordered structure, called concept lattice, from a formal context. We propose to use this method to classify available services of service registry according to user specification.

**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  $A$ . As an example, Table I is an illustration of a formal context with

<sup>1</sup>SODA is a European project partly funded by French Ministry of industry bringing together, among others, Schneider Electric, Thales and Grenoble University.

$O = \{1, 2, 3, 4, 5, 6, 7\}$  and  $A = \{a, b, c, d, e\}$ . A mark in the array means that an attribute is provided by an object.

	a	b	c	d	e
1	X	X	X		
2		X	X	X	
3			X	X	X
4	X	X			
5	X		X		
6				X	
7					X

Table I  
EXAMPLE OF FORMAL CONTEXT.

**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  $I$ . In the previous example (Table I),  $(\{1, 2\}; \{b, c\})$  is a formal concept. Indeed, objects 1 and 2 share the attributes  $b$  and  $c$ . Contrarily,  $(\{1, 2, 3\}; \{c, d\})$  and  $(\{1, 2\}; \{c\})$  are not formal concepts.

By noting  $X$  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:  $(E_1, I_1) <_c (E_2, I_2)$  if  $E_2 \subset E_1$  and  $I_1 \subset I_2$ .

**Definition 3:** A *concept lattice* is defined as the pair  $(\mathcal{C}(\mathbb{K}), \leq_c)$ . It can be represented by a particular graph called Hasse Diagram (Figure 2). 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 ([10]). Most of the time we have  $n \ll m$  and the complexity becomes  $\mathcal{O}(m^2 \times |\mathcal{C}(\mathbb{K})|)$ .

**Definition 4:** We call *top* (resp. *bottom*) the concept whose the intent is equal to the set of all attributes (resp. of all objects). Note that, most of the time, the extent of the top (resp. the intent of the bottom) is the empty set. The top (resp. bottom) is denoted by  $\top$  (resp.  $\perp$ )

**Definition 5:** 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  $(E_1, I_1) <_c (E_2, I_2)$ . Given  $I_1$  a subset of  $A$ , we note by  $successors(I_1)$  the set of successors of the concept  $(E_1, I_1)$ .

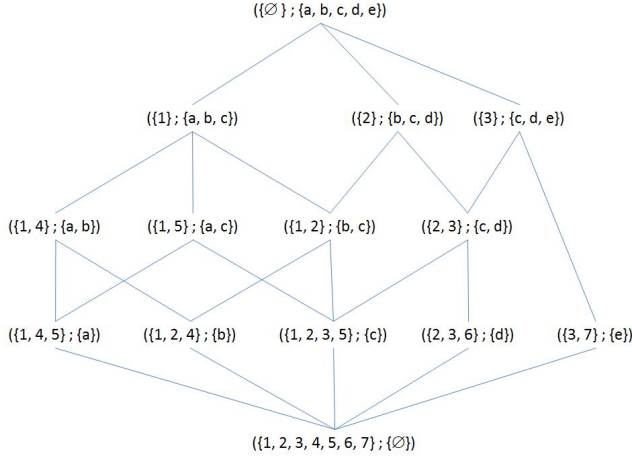


Figure 2. Hasse Diagram of the formal context given in table I.

### III. PROPOSITION

#### A. Principle

Our objective is to help user select services at runtime among the currently available ones. Our approach is illustrated by the Figure 3. First, the user formulates a request for service(s). As a function of the current runtime conditions, a decision tree made of services meeting the request is provided. The user then selects a service in the tree. This approach provides more complete answers to user requests. Instead of getting a single service, the user has a number of possible services related in such a way that they can be easily searched. Also, as we will see in more details, the tree can be used to recover very rapidly from the disappearance of the selected service.

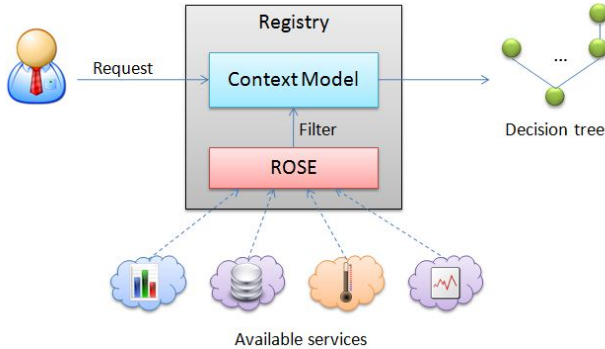


Figure 3. Global approach.

A service request is made of mandatory features and optional features. Mandatory features generally include the services functionalities and a number of important characteristics like security properties. Optional features can include, for instance, a preferred technology or a preferred device manufacturer. Of course, the scope of the mandatory and optional features is very application-specific. Finally,

let us note here that, in this study, we work in well-defined domains. In particular, our industrial partners perfectly know the service interfaces they expect to use. They simply do not know their dynamicity.

The two following sections detail the service registry and the computation of the aforementioned decision tree.

#### B. Service Registry

The aim of the service registry is to maintain a global view of the available services at runtime. Our service registry is divided into two parts: an integration platform named ROSE [11] and a context model.

The ROSE integration platform is an OSGi-based open source middleware<sup>2</sup>. It monitors the runtime environment in such a way that it traces services availability and provides information about them. In pervasive environment, these capabilities are essential because services related to device 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 [12].

Since ROSE detects all the services being in the environment, we have defined a filter to compute application-specific context models. The filter allows to specify the services of interest at diverse levels of abstraction. In fact, only the services of the interest for the application are selected. For instance, a multimedia entertainment application requires multimedia services such as movies library, TV...

We then propose to use the FCA approach, as defined in section II, to organize the filtered services. More precisely, as illustrated by Table II, the context model is a relation between the filtered services ( $s_1, \dots, s_n$ ) and the possible service features in the domain. We categorize the service features into three groups :

- $t$ : The service technologies (WS, UPnP, DPWS...),
- $f$ : The service functionalities,
- $nf$ : 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 II  
CONTEXT MODEL AS A FORMAL CONTEXT.

#### C. Decision Tree

From the context model expressed as a formal context, a concept lattice can be computed. The lattice is composed of a set of concepts that can be classified into two exclusive groups, as illustrated by Figure 4:

- **concepts with no real meaning.** These concepts contain in their intent a set of properties which is not

<sup>2</sup><http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/Rose>

usable. For example, all the concepts with an intent composed of only non-functional properties do not make sense. The *bottom* and the *top* of the lattice are also meaningless. The *bottom* 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 *top* 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.** Contrarily 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.

This classification into concepts with or without applicative meaning is key to our approach. According to concept semantics, we can compute only the interesting concepts and not the entire lattice. The computation of a lattice has a complexity in  $\mathcal{O}(m^2 \times |\mathcal{C}(\mathbb{K})|)$ . The space complexity is in  $\mathcal{O}(2^n)$  since the number of concepts is potentially  $2^n$ . Another limit of the entire lattice use is that we are in a pervasive environment, *i.e.* services appear and disappear. The algorithm for building lattices given in [10] is **incremental** and allows to insert new elements in the lattice of concepts without rebuilding the entire lattice. Unfortunately, deleting an object in the context often implies to compute a new lattice. For this reason it remains difficult to maintain the lattice taking into account the arrival and departure of services over long periods of time.

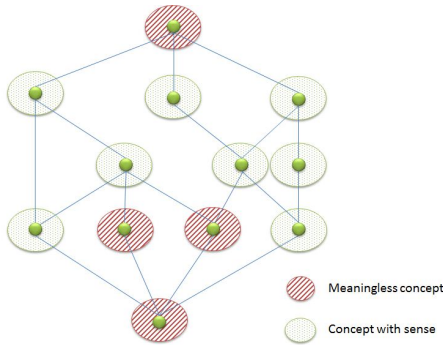


Figure 4. Example of concept classification.

We propose to compute only the interesting concepts meeting a user request (Figure 5). The interesting concepts are a subset of meaningful concepts extracted from the lattice. The subset is a tree where the root element is a formal concept and the nodes are the successors of the formal concept. The successors are ordered as explain in section II.

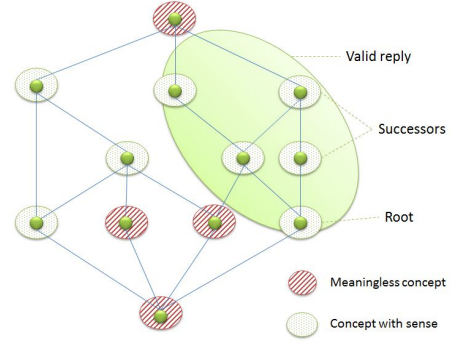


Figure 5. Example of decision tree.

The tree can be viewed as a decision tree which allows to classify and select service according to the user request. In the next section, we present how to compute the root element and how to use the decision tree. Note that the complexity to compute a concept and its successors is  $\mathcal{O}(m^2 \times |successors()|)$ .

#### IV. CLASSIFICATION AND SELECTION

In this section, we present the computation and the use of the decision tree for the different user requests. First, we detail classification and selection for the activities of the workflow defined in the introduction. Then, we explain the impact of the annotations on the dataflow for selection.

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

##### A. From abstract services to concrete services

The classification and the selection of services are detailed according to different criteria described in the user request. The theoretical solution is illustrated by an example. We propose to apply the computation on the extract of context model defined in Table III.

	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	X
$S_5$		X		X	X			X
$S_6$		X		X	X	X		
$S_7$		X		X	X			
$S_8$			X	X				X

Table III  
EXTRACT OF CONTEXT MODEL.

For clarification purposes, the extract of context model contains only the functional attributes *Temperature* and *Humidity* and only the services providing the *Temperature* functionality. Three kinds of selection can be envisaged:

**Selection only based on mandatory features.** In Figure 1, *Temperature* activity is an example of such selection. All the services providing the temperature functionality must be selected. The solution is to compute the formal concept which the intent contains the mandatory features (*MF*) defined in the user request. The mandatory features can be technical, functional and/or non-functional:

$$(S; \mathbf{MF} \cup \mathbf{FF})$$

The sets *S* and *FF* can be empty. If the extent *S* is empty, there is no service available providing the mandatory features. In our example, from the formal context (Table III), the selection result for *Temperature* activity is the formal concept  $(\{S_1, S_2, S_3, S_5, S_6, S_7, S_8\}; \{\mathbf{Temperature}\})$ .

#### Selection based on mandatory and optional features.

This selection is an extension of the previous case. To take into account the optional features, we propose to compute the successors of the concept  $(S; \mathbf{MF} \cup \mathbf{FF})$ . The computation of the successors is an extract of the concept lattice that can be viewed as the decision tree:

$$(S; \mathbf{MF} \cup \mathbf{FF}) \cup \text{successors}(\mathbf{MF} \cup \mathbf{FF})$$

According to the optional features (user preference), the branches can be pruned. The selection is guided by the decision tree. For example, the classification of the *Temperature* services is 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 tree illustrated by Figure 6.

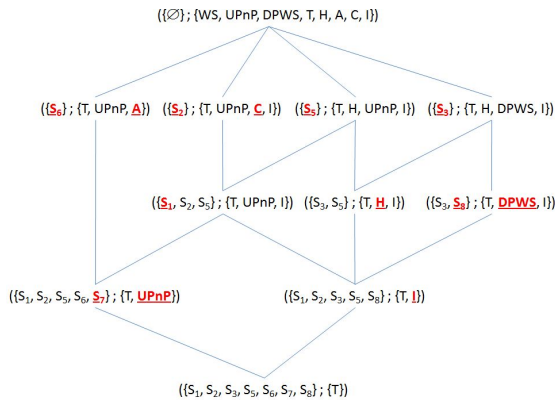


Figure 6. Example of extract lattice for *Temperature* activity.

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. An optional criterion for the user request 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  $(\{S_1, S_2, S_5\}; \{T, UPnP, I\})$ .

To conclude, the user can choose a service according to the optional features guided by the decision tree, the user decides which services are more appropriated to his/her requirements.

**Selection of services meeting the mandatory features with a minimum of additional properties.** The aim of this selection, somehow, is to minimize the side effects. First, as previously, the concept  $(S; \mathbf{MF} \cup \mathbf{FF})$  is computed with the mandatory features as input. To minimize the features of the selected services, we exclude from the set of services *S* all the services that appear in the extent of the successors of the concept  $(S; \mathbf{MF} \cup \mathbf{FF})$ :

$$S' = S \setminus \{\cup_{(S_i, X) \in \text{successor}(\mathbf{MF} \cup \mathbf{FF})} S_i\}$$

For example, the selection of services providing the functionality *Temperature* implemented with UPnP gives the concept  $(\{S_1, S_2, S_5, S_6, S_7\}; \{T, UPnP\})$ . The successors of this concept are  $(\{S_6\}; \{T, UPnP, A\})$ ,  $(\{S_1, S_2, S_5\}; \{T, UPnP, I\})$ ,  $(\{S_2\}; \{T, UPnP, C, I\})$ ,  $(\{S_5\}; \{T, H, UPnP, I\})$  and the *top* (left side of the Figure 6). Consequently, only the service  $S_7$  provides the given properties.

#### B. Impact of annotation on data flow

In this section, we detail the impact of annotations on data flow. In fact, the annotations impact the output and the input of the selected services (Figure 7). We note  $A_1$  the abstract service in input of the dataflow and  $A_2$  the abstract service in output. In our work, we focus on the security constraints [13].

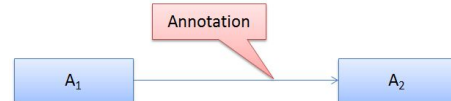


Figure 7. Annotation on data flow.

In this section, we present two selection mechanisms taking annotations into account. We illustrate our results with an example based on the formal context defined in

Table IV. To simplify, the extract of context model contains only the interesting functionalities (*Analysis* and *Storage*) and the possible encryption algorithms<sup>3</sup> (*Triple DES*, *AES 128* and *AES 256*) ensuring confidentiality. Confidentiality is actually a general security concept that can be obtained with encryption algorithms. Let us note here that annotations can be *precise* or *generic* depending on the level of detail they specify (that is encrypted or, more generally, confidential).

	WS	Analysis	Storage	Confidentiality					
				Triple DES input	Triple DES output	AES 128 input	AES 128 output	AES 256 input	AES 256 output
$S_{10}$	X	X			X				
$S_{11}$	X	X					X		
$S_{12}$	X	X							X
$S_{13}$	X		X	X					
$S_{14}$	X		X					X	
$S_{15}$	X	X			X				

Table IV  
EXTRACT OF CONTEXT MODEL.

**Precise annotations on data flow.** In this case, the computation of the formal concept for each abstract service must take into consideration the annotation, *i.e.* the intent must contain the information provided by the annotations. The intent for  $A_1$  is defined by the features of  $A_1$  and the annotation as output:

$$A_1: (S; \mathbf{MF} \cup \mathbf{Annotation\ output} \cup FF) \cup \text{successors}(\mathbf{MF} \cup \mathbf{Annotation\ output} \cup FF)$$

The intent for  $A_2$  is defined by the features of  $A_2$  and the annotation as input:

$$A_2: (S; \mathbf{MF} \cup \mathbf{Annotation\ input} \cup FF) \cup \text{successors}(\mathbf{MF} \cup \mathbf{Annotation\ input} \cup FF)$$

For instance, the communication between the *Analysis* and *Storage* activities is annotated by a confidentiality property made with Triple DES algorithm (Figure 1). The solution of the classification is:  $(\{S_{10}, S_{15}\}; \{WS, Analysis, Triple\ DES\ output\})$  for *Analysis* and  $(\{S_{13}\}; \{WS, Storage, Triple\ DES\ input\})$  for *Storage*.

This type of annotations has a minimal impact on the computation of the appropriate concrete services. Complexity is the same as the one of a formal concept.

**Generic annotations on data flow.** This case is a generalization of the previous case. The intent for  $A_1$  is defined by the features of  $A_1$  and the annotation as output. The intent for  $A_2$  is defined by the features of  $A_2$  and the annotation as input. The difficulty is that the constraint is generic. The

solution proposed must be coherent, *i.e.* the  $A_1$  output must be compatible with the  $A_2$  input.

As previously explained, we propose to compute the formal concept for the mandatory features of  $A_1$  and its successors:

$$A_1: (S; \mathbf{MF} \cup FF) \cup \text{successors}(\mathbf{MF} \cup FF)$$

Then,  $A_2$  is computed according to the annotations found in the intent of the successors of  $A_1$ . First, we compute the Possible Annotation Output denoted by  $PAO = \{a \in Annotations \mid \text{successors}(a) \cap \text{successors}(\mathbf{MF} \cup FF) \setminus \top \neq \emptyset\}$ . More precisely, by noting *Annotations* the set of annotations, providing the annotation functionality, appearing in the service registry and in the set of successors of  $A_2$  ( $\text{successors}(\mathbf{MF} \cup FF)$ ). Then, the computation of  $A_2$  is:

$$A_2: \bigcup_{a \in PAO} (S; \mathbf{MF} \cup a \cup FF) \cup \text{successors}(\mathbf{MF} \cup a \cup FF)$$

This use case is illustrated by Figure 8. For example, if there is an annotation *Confidentiality* between the activities *Analysis* and *Storage*, the set *Annotations* =  $\{TripleDES(input/output), AES128(input/output), AES256(input/output)\}$  defines the encryption algorithms ensuring the confidentiality property defined in the context model (Table IV). The computation for  $A_1$  gives an extract of lattice (left part of Figure 8). The result of the computation for  $A_2$  according to the Possible Annotation Output ( $PAO = \{TripleDESoutput, AES256output\}$ ) is two extracts of lattice (right part of Figure 8).

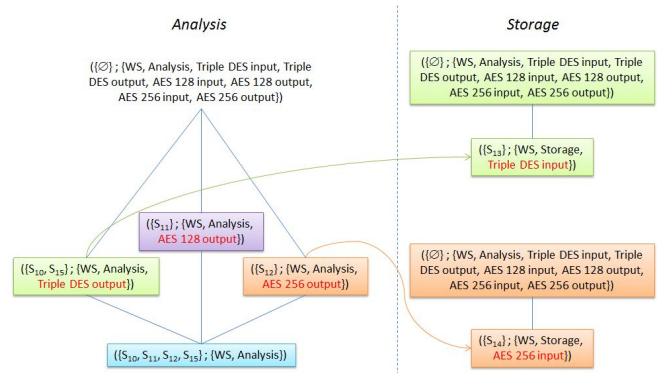


Figure 8. Example of confidentiality annotation.

The result is a graph composed of lattice extracts. If there is a path between the concepts of  $A_1$  lattice extract and  $A_2$  lattice extracts, there is one or more solutions realizing the generic annotation. For example, there is a path between  $(\{S_{10}, S_{15}\}; \{WS, Analysis, TripleDESOutput\})$  and  $(\{S_{13}\}; \{WS, Storage, TripleDESinput\})$ . For the *Analysis* activity, the user can choose the services  $S_{10}$  and/or

<sup>3</sup><http://www.w3.org/TR/xmlenc-core/#sec-Algorithms>

$S_{15}$  and this choice implies that the service  $S_{13}$  must be used to ensure the *Storage* activity.

## V. IMPLEMENTATION

Our approach have been implemented in Java language. In this section, we do not detail the ROSE implementation explained in [11]. The context model is an XML file and its structure (Figure 9) is the following:

- first, the definition of all the attributes, *i.e.* the service properties, defined by a string,
- second, the definition of all the objects, *i.e.* the filtered available services, defined by an identifier and a list of attributes provided and/or required by the services.

```
<!ELEMENT binaryRelation (attributes,objects)+ >
<!ELEMENT attributes attribute+ >
<!ELEMENT attribute (#PCDATA)>
<!ELEMENT objects object+ >
<!ELEMENT object attribute+ >
<!ATTLIST object id ID #REQUIRED >
```

Figure 9. DTD file.

The context model is stored using two *HashMap*. Such a data structure allows the efficient implementation of common operations. Lattices are managed using *JgraphT*<sup>4</sup>. *JgraphT* is a library usually used in graph theory.

The visualization of the decision tree is made with the *JGraph* library<sup>5</sup>. *JGraph* is a library based on *Swing*. Figure 10 is an example of a decision tree computed from a context model containing 50 services and 28 properties.

## VI. DISCUSSION

In this paper, we use the Formal Concept Analysis as a classification tool to select appropriate services according to user specification and a set of available services. It brings the following properties.

**Avoid the selection of no service.** The selection of a service corresponds to the computation of one formal concept and, if necessary, its successors. The formal concept is computed from a set of mandatory features. But, in certain cases, the intent formal concept contains not only the mandatory features, *i.e.* it contains a set of other features. With traditional selection (*e.g.* standard queries in database domain), the query returns an empty set. With FCA, we have not a negative answer. The response is that there is no service exactly providing all the mandatory features but there are services with the mandatory features and with other features. The user must decide if other features can be used to choose among the services of the extent.

**Equivalent services.** For each abstract service, we compute a formal concept or an ordered set of formal concepts.

<sup>4</sup><http://www.jgraph.org/>

<sup>5</sup><http://www.jgraph.com/>

Each formal concept is composed of an extent and an intent. The extent is the set of services that share the features of the intent. If the extent contains more than one service, we can say that these services are equivalent, *i.e.* they have the same characteristics (functional, non-functional and technical). This property is very important in pervasive computing, because the environment is dynamic. When a service departure occurs, the other services of the extent can be used. Consequently, reaction time is reduced: the service registry is queried just one time per activity specification. This search is done in the size of the service registry ( $\mathcal{O}(n \times m)$ ), because we compute only one formal concept.

**Classification of services.** With FCA, services are classified according to a set of optional features defined by the user at specification time. An extract of the concept lattice is computed to classify the services. It can be viewed as a search tree. This classification allows to have a more precise selection of services. It is possible to extend our approach in adding a weight to the branches of the search tree according to the user preferences defined on the optional features.

**Backtracking at runtime.** The equivalent classes of services allow to dynamically adapt the orchestration at runtime for each abstract service. This mechanism is not sufficient to dynamically select the appropriate services. The computation of search tree allows also the adaptation to dynamic environments. The search tree can be used for backtracking at runtime. The search tree for abstract services with optional annotation can be explored according to the availability (departure) of services. This adaptation is also possible for the activities with annotation on output/input dataflow. The consequence of generic annotation on dataflow is to compute a particular graph that can be also viewed as search tree. The advantage of this approach is that the search tree is computed with a complexity  $\mathcal{O}(n^2 \times m)$  and the selection can be made without a new search in the service registry.

## VII. CONCLUSION

Runtime adaptability of pervasive applications built in service-oriented environments largely depends on service selection. In previous works, we observed that, in many industrial use cases, brute force like algorithms for services selection are not effective. They are too costly and not adapted to situations where constraints may be released.

In this paper, we have presented a way to structure services available at runtime based on the Formal Concept Analysis approach. Our purpose is to speed up the selection process and to improve decision making through the building of a concept lattice. The complexity of such computation is in the order of brute force algorithms. But, it can be reused to perform more complex searches, where constraints are changed. In this situation, the equivalence classes and successors avoid reiterate each time the selection algorithm which significantly improves performance at runtime.

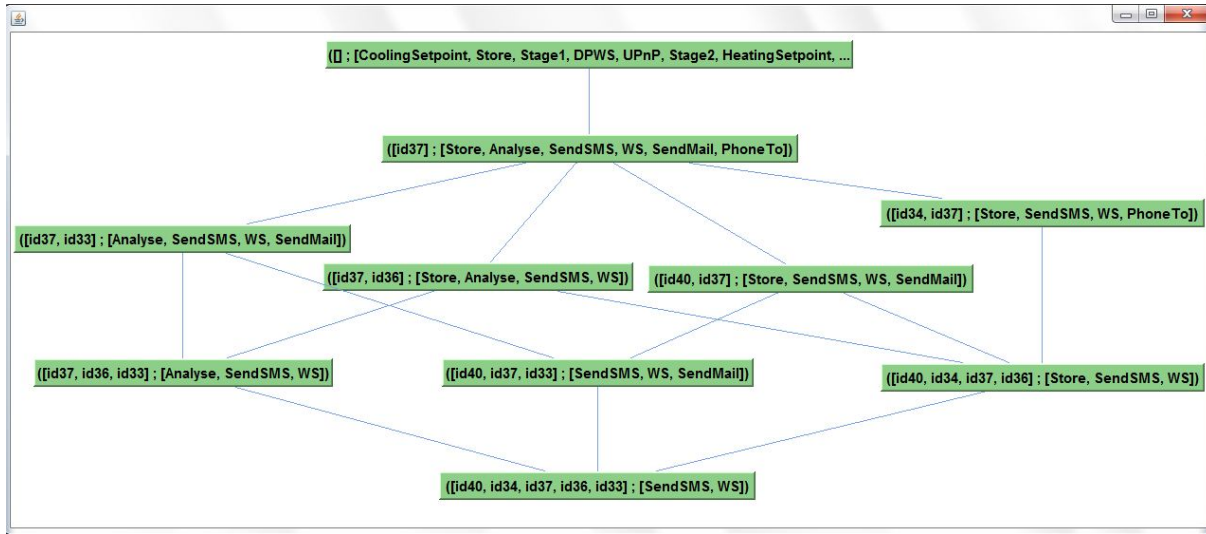


Figure 10. Screenshot of a decision tree.

We are currently integrated this approach more finely with Rose, the extensible framework for the discovery and publishing of resources in service-oriented architecture presented here before.

#### REFERENCES

- [1] Z. Azmeh, M. Huchard, C. Tibermacine, C. Urtado, and S. Vauttier, "WSPAB: A Tool for Automatic Classification & Selection of Web Services Using Formal Concept Analysis," in *European Conference on Web Services (ECOWS 2008)*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 31–40.
- [2] D. Bianchini, V. De Antonellis, B. Pernici, and P. Plebani, "Ontology-based methodology for e-service discovery," *Information Systems - Special issue: The semantic web and web services*, vol. 31, pp. 361–380, June 2006.
- [3] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints," *ACM Transactions on the Web*, vol. 1, Month 2007.
- [4] N. B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny, "QoS-aware service composition in dynamic service oriented environments," in *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. New York, NY, USA: Springer-Verlag, 2009, pp. 1–20.
- [5] G. Canfora, M. D. Penta, R. Esposito, F. Perfetto, and M. L. Villani, "Service Composition (re)Binding Driven by Application-Specific QoS," in *Proceedings of 4th International Conference on Service-Oriented Computing (ICSOC 2006)*, ser. Lecture Notes in Computer Science, vol. 4294. Springer, 2006, pp. 141–152.
- [6] B. Ganter and R. Wille, *Formal Concept Analysis - Mathematical Foundations*. Berlin, Heidelberg: Springer, 1999.
- [7] T. Tilley, R. Cole, P. Becker, and W. P. Eklund, "A Survey of Formal Concept Analysis Support for Software Engineering Activities," in *Formal Concept Analysis*, ser. Lecture Notes in Computer Science, B. Ganter, G. Stumme, and R. Wille, Eds., vol. 3626. Springer, 2005, pp. 250–271.
- [8] G. Stumme, "Efficient Data Mining Based on Formal Concept Analysis," in *Proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA'02)*, ser. Lecture Notes in Computer Science, A. Hameurlain, R. Cicchetti, and R. Traummüller, Eds., vol. 2453. London, UK: Springer-Verlag, 2002, pp. 534–546.
- [9] U. Priss, "Linguistic Applications of Formal Concept Analysis," in *Formal Concept Analysis*, ser. Lecture Notes in Computer Science, B. Ganter, G. Stumme, and R. Wille, Eds., vol. 3626. Springer, 2005, pp. 149–160.
- [10] L. Nourine and O. Raynaud, "A fast incremental algorithm for building lattices," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 14, no. 2-3, pp. 217–227, 2002.
- [11] J. Bardin, P. Lalanda, and C. Escoffier, "Towards an Automatic Integration of Heterogeneous Services and Devices," in *Proceedings of 2010 IEEE Asia-Pacific Services Computing Conference*. Los Alamitos, CA, USA: IEEE Computer Society, 2010.
- [12] P. Lalanda, J. Bourcier, J. Bardin, and S. Chollet, "Development of service-oriented pervasive home applications," in *Smart Home Systems, I*. Book, Ed. Mahmoud A. Al-Qutayri, January 2010.
- [13] S. Chollet and P. Lalanda, "An extensible Abstract Service Orchestration Framework," in *Proceedings of IEEE International Conference on Web Services (ICWS 2009)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2009, pp. 831–838.