Semantic service substitution in pervasive environments

Noha Ibrahim

University of Grenoble,
LIG – Grenoble Informatics Laboratory,
F-38402 St. Martin d'Heres, Grenoble, France
Email: noha.ibrahim@imag.fr

Frédéric Le Mouël* and Stéphane Frénot

University of Lyon,
INSA-Lyon, INRIA CITI Lab,
F-69621 Villeurbanne, France
Email: frederic.le-mouel@insa-lyon.fr
Email: stephane.frenot@insa-lyon.fr
*Corresponding author

Abstract: A computing infrastructure where 'everything is a service' offers many new system and application possibilities. Among the main challenges, however, is the issue of service substitution for the application execution in such heterogeneous environments. An application would like to continue to execute even when a service disappears, or it would like to benefit from the environment by using better services with better QoS when possible. In this article, we define a generic service model and describe the equivalence relations between services considering the functionalities they propose and their non-functional QoS properties. We define semantic equivalence relations between services and equivalence degree between non-functional QoS properties. Using these relations we propose semantic substitution mechanisms upon the appearance and disappearance of services that fit the application needs in a pervasive environment. We developed a prototype as a proof of concept and evaluated its efficiency over a real use case.

Keywords: service-oriented architecture; service substitution; semantic matching; semantic distance; quality of service; z-score; equivalence relations.

Reference to this paper should be made as follows: Ibrahim, N., Le Mouël, F. and Frénot, S. (2014) 'Semantic service substitution in pervasive environments', *Int. J. Services, Economics and Management*, Vol. 6, No. 4, pp.283–309.

Biographical notes: Noha Ibrahim holds a 'Diplôme d'Ingénieur' from the Ecole Nationale Supérieure d'Informatique et de Mathématique Appliquée de Grenoble (ENSIMAG), and a PhD from National Institute for Applied Sciences (INSA Lyon). Her dissertation focused on providing a spontaneous service integration middleware adapted for pervasive environments. She is currently Associate Professor in the Grenoble Informatics Laboratory (LIG) where she works on service composition framework for optimising queries and data mining for multimedia applications.

Frédéric Le Mouël is currently Associate Professor in the National Institute for Applied Sciences of Lyon (INSA Lyon), Telecommunications Department – high-ranked school in France, part of the University of Lyon. He conducts his research in the Center for Innovation in Telecommunication and Integration of Services (INRIA CITI Lab.) where he is leading the Dynamic Software and Distributed Systems for the Internet of Things research group (DynaMid Team). He joined Shanghai Jiao Tong University (SJTU) as Visiting Professor in 2013. His main interests are distributed systems, operating systems, middleware, virtual machines, programming languages, especially in dynamic and autonomic environments.

Stéphane Frénot holds a 'Diplôme d'Ingénieur' from INSA Lyon, and a PhD from University Lyon I about distributed information systems in hospitals. He is currently Professor at the Center for Innovation in Telecommunication and Integration of Services (INRIA CITI Lab.), Telecommunications Department of the National Institute for Applied Sciences of Lyon (INSA Lyon) and in the Rhône-Alpes Complex Systems Institute (IXXI). He is co-heading the INRIA Dice team and is particularly interested in data, web, programming and geopolitics.

1 Introduction

A computing infrastructure (Erl, 2005) where 'everything is a service' offers many new system and application possibilities. Among the main challenges, however, is the issue of service substitution for the application execution in such heterogeneous environments. An application would like to continue to execute even when a service disappears, or it would like to benefit from the services in the environment by using better services with better quality of service when possible.

A service publishes a functional interface, describing all the operations that the service can execute. This description is based on semantics and ontologies (Bittner et al., 2005) as pervasive environments (Weiser, 1991; Satyanarayanan, 2001) are populated with services from different providers and technologies. Besides the semantic interface description, the interface operations have non-functional properties corresponding to their quality of service. Many middleware and architectures proposed solutions for service substitution (Fredj et al., 2008) or service adaptation (Floch, 2006), but very few described by models, definitions and metrics semantic service substitution adapted for pervasive environments and based on functional interface matching and quality of service computing.

The major contributions of the article are in defining and formalising:

- The equivalence relations between services considering the functionalities they propose via their functional interfaces. We define and formalise the service model and the service equivalence relations based on the semantic description of their interfaces and operations. Theses relations allow to define if two services are functionally equivalent or not.
- The QoS degree equivalence functions between the operations and the services. Services can be functionally equivalent but offer and/or require different parameters

for quality of service. The QoS equivalence degree gives a metric that indicates how close two services are in terms of their quality of services.

- Service substitution mechanisms for applications executing in pervasive environments. Based on service equivalence relations and QoS equivalence degree, the pervasive environment can decide to substitute services by functionally equivalent ones, with better QoS computed via the QoS equivalence degree. These service substitution are done transparently and spontaneously as services appear and disappear in the environment with the users coming and leaving.

To define, formalise and explain our relations and metrics we adopted the following model. The 'DEFINITION' paragraphs define the relations and functions between concepts, operations and services using simple grammar and language, whereas the 'EXAMPLE' paragraphs illustrate and explain these definitions via a use case.

We begin in Section 2 by exposing the state of the art. We define in Section 3 the service equivalence relations and the non-functional QoS degree equivalence metrics. We then explain in Section 4 the semantic service substitution in pervasive environments. Section 5 details the developed proof-of-concept prototype and its results. Finally, Section 6 concludes and gives perspectives for this work.

2 State of the art

In pervasive environments, service substitution (Fredj et al., 2008; Santhanam et al., 2009) and service similarity problems (Kokash, 2006; Mokhtar, 2007; Aït-Bachir and Fauvet, 2009) have become the new trends in the service-oriented community after the service discovery and service composition problems. Once services are deployed, accessed, executed and composed, the pervasiveness of the environment imposes researchers to find solutions for the service unavailability problem. Indeed, in a pervasive environment, services can come and go without prior notification and finding the right substitute for a given service is very often a hard task to achieve. In the literature, we distinguish three types of service similarity: those concerning the structural part or functional property of services, those concerning the behavioural part of services and those that deal with the non-functional properties of services.

Structural similarity between services (Kokash, 2006) is a functional matching algorithm between the interface WSDL descriptions of web services. The algorithm takes the description of web services and is able to tell if the two services are similar using a semantic similarity metric. But this work need to be optimised and especially the non-functional parts of services need to be taken into account. Perse (Mokhtar, 2007) proposes a QoS metric for web services based on normalisation functions but this metric is used to dynamically compose services together. Perse does not consider service substitution as a separate problem from service composition. Finally, EurekaBESERIAL (Aït-Bachir and Fauvet, 2009) proposes an algorithm that is capable of detecting all the incompatibilities between two interface behaviour for web services and based on these incompatibilities it introduces a similarity function to compare two web services behaviour but it does not take into account the non-functional properties of services.

Some works deal with service substitution (Fredj et al., 2008; Santhanam et al., 2009). Siroco (Fredj et al., 2008) proposes a framework that substitutes stateful web services, taking into account the state of a service when executing and ensuring to applications a service continuity when substituting the service. But Siroco does not deal for now with non-functional properties. Santhanam et al. (2009) propose a web service substitution based on preferences over non-functional attributes but their description of non-functional properties is not general enough to take all types of non-functional properties into consideration. In this article, we do not limit our model to web services as all major systems do but propose a general model of a service, describing its functional and non-functional properties (quantitative or qualitative) and based on this model we propose different metrics for computing non-functional service similarities. Than, we propose a mechanism for service substitution that substitute services not only upon their unavailability as the major systems do, but also when a new service fits better an application.

3 Service functional and non-functional QoS equivalence relations

3.1 Service model

We define a generic service model as composed of a functional interface and non-functional QoS properties. A functional interface specifies operations that can be performed on the service. An operation is described by a concept, a set of inputs and an output. The QoS non-functional properties describe the operation capabilities. These capabilities reflect the quality of the functionality expected from the service, such as dependability (including availability, reliability, security and safety), accuracy of the operation, speed of the operation, and so on. The service is also semantically described. The semantic description is upon the operations and QoS properties and is based upon common ontology concepts.

Consider finite sets of grammatical alphabet Σ, ontologies O, concepts N belongings to these ontologies O, operations Op, inputs In, outputs Out, concepts Cpt, non-functional properties Np, quantitative and qualitative non-functional properties Np_{QN}, Np_{QL}. Consider the following operators: * (repetition zero or more times), + (repetition one or more times), | | (the number of occurrences) and $^{0..1}$ (repetition zero or one time).

We define an operation op belonging to $Op \subset \mathrm{Op}$ as follows:

$$\big(op \in Op \Leftrightarrow \exists In \subset \mathrm{In},\ \exists Out \subset \mathrm{Out},\ \exists cpt \in \mathrm{Cpt},\ \exists Np \subset \mathrm{Np} \subset,\ \exists Np_{QN} \subset Np_{QN},\ \exists Np_{QL} \subset Np_{QL}\big):$$

$$op :< In^*,\ Out^{0..1},\ cpt,\ Np^* >$$

$$in :< name, type, semantic >,\ name \in \Sigma^*$$

$$cpt :< name, semantic >,\ name \in \Sigma^*$$

$$type :< language, name >,\ \{name, language\} \in \Sigma^*$$

$$semantic :< o, n >, o \in O \subset \mathrm{O},\ n \in N \subset \mathrm{N}$$

$$np :< Np_{QL}^{*}, Np_{QN}^{*} >$$

$$np_{QL} :< name, semantic >, \ \ name \in \Sigma^*$$

$$np_{QN} :< name, numericValue, operator >, \ \ name \in \Sigma^*$$

$$numericValue \in \mathbb{R}$$

$$operator : \{<, \ >, \ \leq, \ \geq\}$$

where

- In is the set of the operation op inputs. *in* is defined as a tuple where *name* is the chosen input syntactic name, *type* is the syntactic input type, and semantic the input *semantic* description.
- $out \in Out$ is the operation *op* output. *out* is defined as a tuple where *type* is the output syntactic type, and *semantic* its semantic description.
- *cpt* is the concept the operation *op* defines. The operation *op* concept *cpt* is defined as a tuple, where *name* is the syntactic name through which the operation is called and *semantic* its semantic description.
- *Np* is the set of non-functional properties characterising *op*. *Np* can be qualitative or quantitative. $np_{QN} \in Np_{QL}$ is the qualitative non-functional properties defined as a tuple <*name*, *semantic*>. $np_{QN} \in Np_{QN}$ is the quantitative non-functional properties defined as a tuple, where $numeriValue \in \mathbb{R}$ and $operator \in \{>, \ <, \ \leq, \ \geq\}$. *operator* specifies the order applied to *numericValue*. For $\{>, \geq\}$ the greater the *numericValue* is, the best is the QoS property for the service runtime execution. For $\{<, \leq\}$ the smaller the *numericValue* is, the best is the QoS property for the service runtime execution.

The *type* depends strongly on the programming language the *op* is defined in, whereas the *semantic* is independent of the technology and more related to the set of defined ontologies *O*.

Our service model is general enough to respect the SOA specifications, and to offer a common model to the heterogeneous technologies usually available in pervasive environments. The model proposes semantic descriptions relying on common ontologies, and by that it allows to abstract from the programming languages.

Example 1: We consider three operations (c.f., Figure 1) and three interfaces (c.f., Figure 2) described under the generic service model. Each operation has a set of inputs described by a name, a type, and a semantic description, an output described by a type and a semantic description, and a concept described by a name and a semantic concept. Each operation can have one or several non-functional properties, qualitative or quantitative. These three operations (c.f., Figure 1) and three interfaces (c.f., Figure 2) are used in the following examples to illustrate the upcoming definitions.

Figure 1 Three operation specifications (see online version for colours)

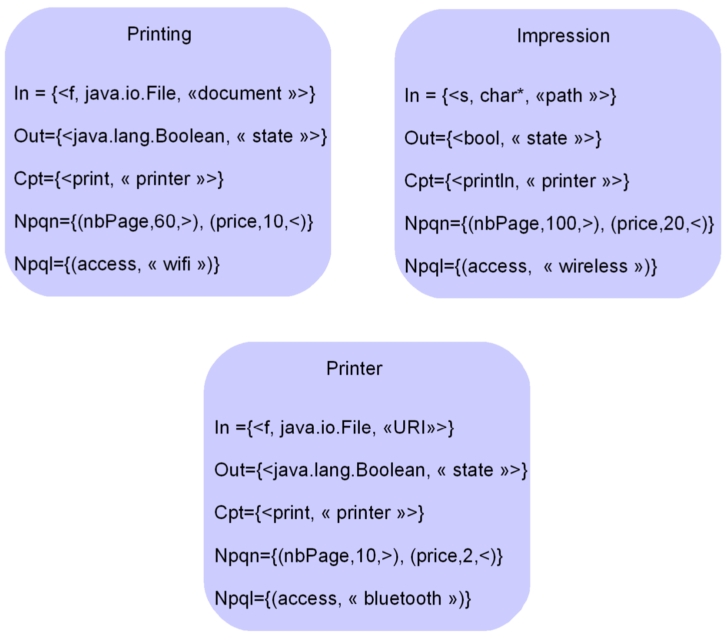
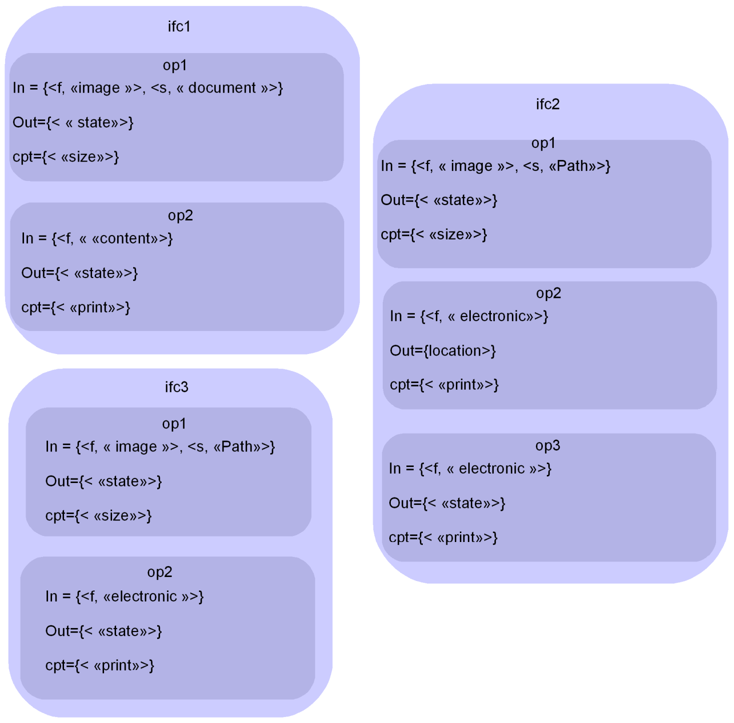

Printing
In = {<f, java.io.File, «document »>}
Out={<java.lang.Boolean, « state »>}
Cpt={<print, « printer »>}
Npqn={(nbPage,60,>), (price,10,<)}
Npql={(access, « wifi »)}
Impression
In = {<s, char*, «path »>}
Out={<bool, « state »>}
Cpt={<println, « printer »>}
Npqn={(nbPage,100,>), (price,20,<)}
Npql={(access, « wireless »)}
Printer
In ={<f, java.io.File, «URI»>}
Out={<java.lang.Boolean, « state »>}
Cpt={<print, « printer »>}
Npqn={(nbPage,10,>), (price,2,<)}
Npql={(access, « bluetooth »)}

Figure 2 Three interface specifications (see online version for colours)



3.2 Service equivalence relations

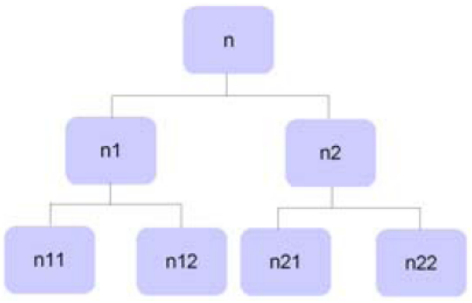
Service equivalence relations determine whether two services offer the same functionality or not. A service is considered equivalent to another one if it can offer the same functionality (same interface) even with different non-functional QoS properties. The aim of this section is to provide definitions of possible relations between services in order to identify and decide when a service can be replaced by another one. Two relations are introduced: the equivalence ($\equiv$) and the almost equivalence ($\triangleright$) relations. In an equivalence relation, the two equivalent entities can interchange and be replaced one by the other. The equivalence relation is reflexive, symmetric, and transitive. In an almost equivalence relation only one entity can replace the other one. This relation is non-reflexive, asymmetric, and transitive. It is based on sub-concept relations in the ontologies used to describe services of the environments. The relations tackle two main parts of a service: its functional interface and its non-functional QoS properties. In the rest of this section, we define our interface equivalence relations and our non-functional QoS equivalence degree.

We define the interface equivalence $\equiv_{sem}$ upon the operation equivalence which itself is defined upon a concept matching M_{Cpt} with concepts belonging to a defined ontology. We begin by defining the concept matching of a given ontology.

3.2.1 Concept matching

The matching of two concepts belonging to the same ontology has been widely studied. We define our matching relation M_{Cpt} between concepts belonging to the same ontology. A concept n belonging to an ontology o (Figure 3), can provide all its immediate sub-concepts n_1 and n_2 or one of its sub-concepts n_1 or n_2. This distinction depends strongly on the ontology definitions and providers. Some research such as Paolucci et al. (2002) made the assumption that by selecting a concept n, we implicitly suppose that it provides all its immediate sub-concepts, others made the other assumption that by selecting a concept n, it provides at least one of its immediate sub-concepts, but not necessarily all of them. Consider the set $\{n_1, n_2, \ldots, n_n\}$ of all the sub-concepts of a concept n in an ontology o, the assumption of Paolucci et al. (2002) is formalised as follows: $n \equiv_{provide} (n_1 \wedge n_2 \wedge \ldots \wedge n_n)$ which means that n can replace n_1, n_2, etc. Others, do not make strong assumptions as this and suppose that a concept n provides one or more of its sub-concepts but not necessarily all of them, $n \equiv_{provide} (n_1 \vee n_2 \vee \ldots \vee n_n)$. We fall into the first category, stipulating that a super-concept offers what its sub-concepts offer, and hence can replace them.

Figure 3 An ontology example (see online version for colours)



Defining *n* and *m*, two concepts belonging to the same ontology *o*. We define the four values of concept matching M_{Cpt} inspired from Paolucci et al. (2002) as follows:

Definition 1: $M_{Cpt}(n, m) = Exact$

If *n* and *m* are equivalent concept.

Definition 2: $M_{Cpt}(n, m) = PlugIn$

If *n* is a super-concept of *m*.

Definition 3: $M_{Cpt}(n, m) = Subsume$

If *n* is a sub-concept of *m*.

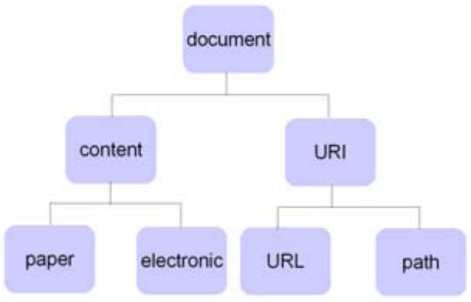
Definition 4: $MCpt(n, m) = Fail$

If *n* and *m* do not verify the above conditions.

Example 2: Using our ontology example Figure 4, we give an example of M_{Cpt}.

$$Example \left| \begin{array}{l} M_{Cpt}("content","electronic") = PlugIn \\ M_{Cpt}("document","URL") = PlugIn \\ M_{Cpt}("paper","document") = Subsume \\ M_{Cpt}("content","path") = Fail \end{array} \right.$$

Figure 4 A document ontology example (see online version for colours)



These concept matching values are the metrics employed to match operations and interfaces of services. We first define the values that the matching of operations can take, and based on these values we define when two operations are equivalent or almost equivalent.

3.2.2 Semantic operation equivalence

Definition 5: Comparable operations $\propto$.

We define two operations *opi* and *opj* to be comparable $(\propto (opi, opj) = true)$ if they have the same number of inputs and the same number of outputs and if it exists a bijection *f* over their inputs allowing to compare the inputs parameters two by two.

$$\forall k, I \in \left\{1 \ldots \left|In_{opi}\right|\right\}$$

$$\begin{aligned}&\left|In_{opi}\right| = \left|In_{opj}\right| \wedge \left(\left|Out_{opi}\right| = \left|Out_{opj}\right|\right)\\ &\wedge \left(\exists f : n_{opi} \rightarrow In_{opj}, \forall n_{opj}, \exists ! ink \in In_{opi}, f\left(in_k\right) = in_l\right)\end{aligned}$$

$\forall \{i, j, l, k\} \in \mathbb{N}$, we define the semantic matching, $M_{sem}(op_i, op_j)$, of two comparable operations op_i and op_j $(\propto (op_i, op_j) = true)$, considering the semantic matching of their concepts, inputs and outputs.

We can quickly realise that the semantic matching of these three items – inputs, outputs, and concepts – can be different, as the concept matching can take multiple values. In a semantic matching, the three items can range from *Exact* matching to *Fail* passing by the *PlugIn* and *Subsume* values.

We define the different values a semantic matching M_{sem} between two operations op_i and op_j can take as follows:

Definition 6: $M_{sem}(op_i, op_j) = Exact$

Two operations op_i and op_j verifying $(\propto (op_i, op_j) = true)$ are *Exact* semantic matching if all the matching values between concept, inputs and output are *Exact*.

$$\begin{aligned}&\forall k \in \mathbb{N} : \left(M_{Cpt}\right)\left(sem_{cpt_{op_i}}, sem_{cpt_{op_j}}\right) = Exact\\ &\wedge\left(\forall in_k \in In_{op_i}, M_{Cpt}\left(sem_{in_k}, f\left(sem_{in_k}\right)\right) = Exact\right)\\ &\wedge\left(M_{Cpt}\left(sem_{out_{op_i}}, sem_{out_{op_j}}\right) = Exact\right)\end{aligned}$$

Definition 7: $M_{sem}(op_i, op_j) = PlugIn$

They are *PlugIn* semantic matching if they are not *Exact* matching and all the matching between concept, inputs or output values are *Exact* or *PlugIn*.

$$\begin{aligned}
&\forall k \in \mathbb{N}: \\
&M_{sem}\left(op_i, op_j\right) \neq Exact \\
&\wedge\left(M_{Cpt}\left(sem_{cpt_{op_i}}, sem_{cpt_{op_j}}\right) \in \{Exact \vee PlugIn\}\right) \\
&\wedge\left(\forall in_k \in In_{op_i}, M_{Cpt}\left(sem_{in_k}, f\left(sem_{in_k}\right)\right) in \{Exact \vee PlugIn\}\right) \\
&\wedge\left(M_{Cpt}\left(sem_{out_{op_i}}, sem_{out_{op_j}}\right) \in \{Exact \vee PlugIn\}\right)
\end{aligned}$$

Definition 8: $M_{sem}(op_i, op_j) = Subsume$

They are *Subsume* semantic matching if they are no *Exact* or *PlugIn* matching and at least one matching value between concept, inputs or output is *Subsume* and no *Fail* matching value is found between outputs, concepts, and the corresponding comparable inputs.

$$\begin{aligned}
&\forall k \in \mathbb{N}: \\
&M_{sem}\left(op_i, op_j\right) \neq Exact \\
&\wedge\left(M_{sem}\left(op_i, op_j\right) \neq PlugIn\right) \\
&\wedge\left(M_{Cpt}\left(sem_{cpt_{op_i}}, sem_{cpt_{op_j}}\right) = \neg(Fail)\right) \\
&\wedge\left(\forall in_k \in In_{op_i}, M_{Cpt}\left(sem_{in_k}, f\left(sem_{in_k}\right)\right) = \neg(Fail)\right) \\
&\wedge\left(M_{Cpt}\left(sem_{out_{op_i}}, sem_{out_{op_j}}\right) = \neg(Fail)\right)
\end{aligned}$$

Definition 9: $M_{sem}(op_i, op_j) = Fail$

They are *Fail* semantic matching if they have different inputs or outputs numbers or at least one semantic matching value between concepts, inputs or outputs is *Fail*.

$$\begin{aligned}
&\forall \{k, l\} \in \mathbb{N}: \\
&\left(\left|In_{op_i}\right| \neq \left|In_{op_j}\right|\right) \\
&\vee\left(\left|Out_{op_i}\right| \neq \left|Out_{op_j}\right|\right) \\
&\vee\left(M_{Cpt}\left(sem_{cpt_{op_i}}, sem_{cpt_{op_j}}\right) = Fail\right) \\
&\vee\left(\exists in_k \in In_{op_i}, \forall in_l \in In_{op_j}, M_{Cpt}\left(sem_{in_k}, sem_{in_l}\right) = Fail\right) \\
&\vee\left(M_{Cpt}\left(sem_{out_{op_i}}, sem_{out_{op_j}}\right) = Fail\right)
\end{aligned}$$

Example 3: Considering the three operations defined in Figure 1.

The semantic matching between these operations give the following values:

$$M_{Cpt} = (Printing, Impression) = PlugIn$$
$$M_{Cpt} = (Printing, Printer) = PlugIn$$
$$M_{Cpt} = (Impression, Printer) = Subsume$$
$$M_{Cpt} = (Impression, Printing) = Subsume$$
$$M_{Cpt} = (Printer, Printing) = Subsume$$
$$M_{Cpt} = (Printer, Impression) = PlugIn$$

The semantic operation matching provides the tools to define when operations are equivalent or almost equivalent.

Definition 10: Operation equivalence.

We define two operations op_i and op_j to be semantically equivalent $\equiv_{sem}$ if:

$$\left(\equiv_{sem}\left(op_i, op_j\right) = true\right) \Leftrightarrow \left(M_{sem}\left(opi, opj\right) = Exact\right)$$

The operation equivalence $\equiv_{sem}$ is reflexive, symmetric, and transitive. We notify that the semantic equivalence satisfies the conditions an equivalence relation $\Re$ needs to fulfil.

Definition 11: Operation almost equivalence.

We define two operations op_i and op_j to be semantically almost equivalent $\triangleright_{sem}$ if:

$$\left(\triangleright_{sem}\left(op_i, op_j\right) = true\right) \Leftrightarrow \left(M_{sem}\left(op_i, op_j\right) - PlugIn\right)$$

The almost equivalence is non-reflexive, asymmetric, and transitive. This relation of almost equivalence specifies that op_i is equivalent to op_j and can replace it but that the contrary is not true. op_j can not always replace op_i.

Example 4: Coming back to our example in Figure 1, where we had these matching values between the three operations Printing, Impression, and Printer:

$$M_{Cpt} = (Printing, Impression) = PlugIn$$
$$M_{Cpt} = (Printing, Printer) = PlugIn$$
$$M_{Cpt} = (Impression, Printer) = Subsume$$
$$M_{Cpt} = (Impression, Printing) = Subsume$$
$$M_{Cpt} = (Printer, Printing) = Subsume$$
$$M_{Cpt} = (Printer, Impression) = PlugIn$$

We can conclude the following almost equivalent relations:

$$\triangleright (Printing, Impression) = true$$
$$\triangleright (Printing, Printer) = true$$
$$\triangleright (Printer, Impression) = true$$

Now, that we have defined the operation equivalence relations, we define the interface equivalence relations and by that we define when two services are equivalent or almost equivalent.

3.2.3 Interface equivalence

We define two interfaces to be comparable $(\propto (ifc_i, ifc_j) = true)$ if they have the same number of operations and if it exists a bijection *f* over their operations allowing to compare them two by two:

Definition 12: Comparable interfaces $\propto$

We define two interfaces *ifc*$_i$ and *ifc*$_j$ to be comparable $(\propto (ifc_i, ifc_j) = true)$ if:

$$\left|Op_{ifc_i}\right| = \left|Op_{ifc_j}\right| \\ \wedge \left(\exists f : Op_{ifc_i} \rightarrow Op_{ifc_j}, \forall op_l \in Op_{ifc_j}, \exists! op_k \in Op_{ifc_i}, f\left(op_k\right) = op_l\right)$$

As for operations we define the semantic matching between two interfaces *ifc*$_i$ and *ifc*$_j$:

Definition 13: $M_{sem}(ifc_i, ifc_j) = Exact$

Two interfaces *ifc*$_i$ and *ifc*$_j$ are Exact semantic match if $\propto (ifc_i, ifc_j) = true$ and:

$$\forall op_i \in Op_{ifc_i}, M_{sem}\left(op_i, f\left(op_i\right)\right) = Exact$$

Definition 14: $M_{sem}(ifc_i, ifc_j) = PlugIn$

They are *PlugIn* semantic match if $\propto (ifc_i, ifc_j) = true,$ and:

$$M_{sem}\left(ifc_i, ifc_j\right) \neq Exact \\ \wedge \left(\forall op_i \in Op_{ifc_i}, M_{sem}\left(op_i, f\left(op_i\right)\right) \in \left\{Exact \vee PlugIn\right\}\right)$$

Definition 15: $M_{sem}(ifc_i, ifc_j) = Subsume$

They are *Subsume* semantic match if $\propto (ifc_i, ifc_j) = true,$ *ifc*$_i$ and *ifc*$_j$ are not *Exact* nor *PlugIn* semantic match and:

$$M_{sem}\left(ifc_i, ifc_j\right) \neq Exact \\ \wedge \left(M_{sem}\left(ifc_i, ifc_j\right) \neq PlugIn\right) \\ \wedge \left(\forall op_i \in Op_{ifc_i}, M_{sem}\left(ifc_i, ifc_j\right)\right) \in \{Exact \vee PlugIn \vee Subsume\}\Big)$$

Definition 16: $M_{sem}(ifc_i, ifc_j) = Fail$

They are *Fail* semantic match if:

$$\propto \left(ifc_i, ifc_j\right) = false \\ \vee \left(\exists\, op_i \in Op_{ifc_i}, \forall op_j \in Op_{ifc_j}, M_{sem}\left(op_i, op_j\right) = Fail\right)$$

It is sufficient to have only one operation op_i of ifc_i that do *Fail* match with any operation op_j of ifc_j to declare that the two services matching fails.

Based on these interface semantic matching definitions, we define the interface equivalence and almost equivalence.

Definition 17: Interface equivalence.

We define two interfaces ifc_i and ifc_j to be semantically equivalent $\equiv_{sem}$ if:

$$\left(\equiv_{sem}\left(ifc_i, ifc_j\right) = true\right) \Leftrightarrow \left(M_{sem}\left(ifc_i, ifc_j\right) = Exact\right)$$

The equivalence $\equiv_{sem}$ is reflexive, symmetric, and transitive.

Definition 18: Interface almost equivalence.

We define two services ifc_i and ifc_j to be semantically almost equivalent $\triangleright_{sem}$ if:

$$\left(\triangleright\, sem\left(ifc_i, ifc_j\right) = true\right) \Leftrightarrow \left(M_{sem}\left(ifc_i, ifc_j\right) = PlugIn\right)$$

As for operations, the almost equivalence is non-reflexive, non-symmetric, and transitive. This relation of almost equivalence specifies that ifc_i is equivalent to ifc_j and can replace it but that the contrary is not true. ifc_j cannot always replace ifc_i.

Example 5: Considering the three interfaces and their semantic descriptions in Figure 2:

The semantic matching between their different operations gives the following values:

$$\begin{aligned}
&\propto\left(ifc1, ifc3\right) = true \\
&\wedge\left(M_{sem}\left(op1_{ifc1}, op1_{ifc3}\right) = PlugIn\right) \\
&\wedge\left(M_{sem}\left(op2_{ifc1}, op2_{ifc3}\right) = PlugIn\right) \\
&We\ can\ implies \Rightarrow \left(\triangleright_{sem}\left(ifc1, ifc3\right) = true\right)
\end{aligned}$$

The two interfaces *ifc*1 and *ifc*2 are not comparable as they do not have the same number of operations. Nevertheless, some of their operations are *PlugIn* semantic.

Many services do not have the same number of operations per interface as depicted in Example 5. To resolve this issue brought by the example. We define the matching over a set of operations for two interfaces ifc_i and ifc_j.

Definition 19: $M_{sem}^{Op}(ifc_i, ifc_j) = Exact$

Two interfaces ifc_i and ifc_j are *Exact* semantic matching over a subset of operations *Op*, if:

$$\begin{aligned}
&\propto^{Op}\left(ifc_i, ifc_j\right) = true \\
&\wedge\left(Op \subset Op_{ifc_i}, \forall opi \in Op, M_{sem}\left(op_i, f\left(op_i\right)\right) = Exact\right)
\end{aligned}$$

Definition 20: $M_{sem}^{Op}\left(ifc_i, ifc_j\right) = PlugIn$

Two services ifc_i and ifc_j are *PlugIn* semantic matching over a subset of operations *Op*, if:

$$\propto^{Op}\left(ifc_i, ifc_j\right) = true$$
$$M_{sem}^{Op}\left(ifc_i, ifc_j\right) \neq Exact$$
$$\wedge\left(Op \subset Op_{ifc_i}, \forall opi \in Op, M_{sem}\left(op_i, f\left(op_i\right)\right) \in \{Exact \vee PlugIn\}\right)$$

We thus define interface equivalence and almost equivalence between interfaces over a subset of operations:

Definition 21: Interface equivalence over a subset of operations, $\equiv_{sem}^{Op}$

We define two interfaces ifc_i and ifc_j to be semantically equivalent over a subset of operations Op:

$$\left(\equiv_{sem}^{Op}\left(ifc_i, ifc_j\right) = true\right) \Leftrightarrow \left(M_{sem}^{Op}\left(ifc_i, ifc_j\right) = Exact\right)$$

Definition 22: Interface almost equivalence over a subset of operations, $\triangleright_{sem}^{Op}$

We define two services ifc_i and ifc_j to be semantically almost equivalent over a subset of equivalence Op:

$$\left(\triangleright_{swm}^{Op}\left(ifc_i, ifc_j\right) = true\right) \Leftrightarrow \left(M_{sem}^{Op}\left(ifc_i, ifc_j\right) = PlugIn\right)$$

Example 6: Coming back to our example in Figure 2. The semantic matching between the different operations of $ifc1$ and $ifc2$ gives the following values:

$$\propto^{op1_{ifc1}, op2_{ifc1}}\left(ifc1, ifc2\right) = true$$
$$\wedge\left(M_{sem}\left(op1_{ifc1}, op1_{ifc2}\right) = PlugIn\right)$$
$$\wedge\left(M_{sem}\left(op2_{ifc1}, op3_{ifc2}\right) = PlugIn\right)$$

From these matching values, we can implies $\Rightarrow (\triangleright_{sem}^{\{op1\}i_{fc1}, op2_{ifc1}\}}(ifc1, ifc2) = true)$

The two interfaces $ifc1$ and $ifc2$ are almost equivalent upon the two operations of $ifc1$.

This equivalence and almost equivalence over subsets of operations is useful for service substitution issues, as a service can be replaced by another one if certain operations are specified to be required by applications at a given time. Many services can be almost equivalent and we need to be able to rank between these almost equivalence relations. A ranking of the semantic matching values need to be introduced. This ranking will help ordering services that have semantic almost equivalence with different concept values for the respective operations' inputs, outputs and concepts. It is also used to rank interfaces and operations that have *Subsume* semantic matching. This operations' ordering allows users and applications to choose services that best suit their requirements at a given time, and re-adapt their choice if other services that have a closer semantic equivalence appear. We introduce a semantic distance D_{sem} between two interfaces. It calculates the distance between two interfaces semantic descriptions. The more this value is closer to zero the more these two services are equivalent.

3.2.4 Semantic distance

Definition 23: Concept semantic distance.

We first define a normalised concept distance D_{Cpt} between two concepts *n* and *m*:

$$D_{Cpt}(n,m): \begin{array}{ll} 0 & \textit{if } M_{Cpt}(n,m) = \textit{Exact} \\ 0.2 & \textit{if } M_{Cpt}(n,m) = \textit{PlugIn} \\ 0.8 & \textit{if } M_{Cpt}(n,m) = \textit{Subsume} \\ 1 & \textit{if } M_{Cpt}(n,m) = \textit{Fail} \end{array}$$

The closer the distance is to zero, the best is the semantic value matching between two concepts. An *Exact* value is preferred to a *PlugIn* one, which is preferred to a *Subsume* one. The choice of values can vary. The idea is to assign different values and especially values that reflect the importance of the matching result. In this definition we chose to distinguish to *Exact* and *PlugIn* from *Subsume* and *Fail*. Other values more ponderated can be chosen.

Definition 24: Operation semantic distance.

We define the semantic distance between two comparable operations *opi* and *opj* $(\propto (opi, opj) = true) : (D_{sem}(opi, opj), i, j \in \mathbb{N})$. This semantic distance is the sum of the ponderated concept distance of the operation concept, inputs and output semantic description:

$$w_1 * D_{Cpt}\left(sem_{cptopi}, sem_{cptopj}\right) + w_2 * D_{Cpt}\left(sem_{out_{opi}}, sem_{out_{opj}}\right)$$
$$+\sum_{k=1}^{|In_{opi}|}\left(w_k * D_{Cpt}\left(sem_{ink_{opi}}, sem_{f\left(ink_{opi}\right)}\right)\right)$$

where $\sum_{i \in \mathbb{N}}(wi) = 1$

w_i corresponds to the weight we wish to give to the concept, inputs and output. When matching two operations, the focus may be put on inputs, outputs parameters or on the concept. w_i allows to ponderate the ranking of operations.

Definition 25: Interface semantic distance.

The semantic distance between two comparable interfaces $(D_{sem}(ifc_i, ifc_i)i, j \in \mathbb{N})$ is the sum of all the semantic distance between their comparable operations, ponderated by a weight allowing to focus on some operations rather than others.

$$\sum_{k=1}^{|Op_{ifc_i}|}\left(w_k * D_{sem}\left(opk_{ifc_i}, f\left(opk_{ifc_i}\right)\right)\right)$$

Example 7: We come back to our example and calculate the semantic distance D_{sem} of our three operations:

$$D_{Cpt}("printer","printer") = 0$$
$$D_{Cpt}("document","URI") = 0.2$$
$$D_{Cpt}("state","state") = 0$$
$$=> D_{sem}(printing, printer) = w2 * 0.2$$

$$D_{Cpt}("printer","printer") = 0$$
$$D_{Cpt}("document","path") = 0.2$$
$$D_{Cpt}("state","state") = 0$$
$$=> D_{sem}(printing, impression) = w2 * 0.2$$

The operations *Printing* is *PlugIn* matching with *Impression* and *Printer* and has the same semantic distance value to both operations.

$$D_{Cpt}("printer","printer") = 0$$
$$D_{Cpt}("pat","document") = 0.8$$
$$D_{Cpt}("state","state") = 0$$
$$=> D_{sem}(impression, printing) = w2 * 0.8$$

The overall value of D_{Cpt}(*printing*, *impression*) < D_{Cpt}(*impression*, *printing*) and is normal as printing is *PlugIn* of *Impression* and *Impression Subsume* of *Printing* (see Example 3).

Example 8: We come back to our example and calculate the semantic distance D_{sem} of our interfaces:

$$D_{Cpt}\left(op1_{ifc1}, op1_{ifc3}\right) = 0.2$$
$$D_{Cpt}\left(op2i_{ifc1}, op2_{ifc3}\right) = 0.2$$
$$=> D_{sem}(ifc1, ifc3) = w1 * 0.2 + w2 * 0.2$$

$$D_{Cpt}\left(op1_{ifc1}, op1_{ifc2}\right) = 0.2$$
$$D_{Cpt}\left(op2i_{ifc1}, op3_{ifc2}\right) = 0.2$$
$$=> D_{sem}(ifc1, ifc2) = w1 * 0.2 + w2 * 0.2$$

The overall values of D_{Cpt}(*ifc*1, *ifc*3) and *DCpt*(*ifc*1, *ifc*2) depends on the values assigned to the weights (*w*1 and *w*2).

In our semantic distance calculation example, we gave the three items of an operation – inputs, output, and concept – the same importance. We can ponderate the semantic distance by introducing weights to each of the operation items.

The equivalences introduced so far concern the interfaces of services. If two services can publish the same interface, they can provide different non-functional properties. If we are able to distinguish services by their functionalities, it is interesting to evaluate how equivalent services are in terms of non-functional QoS properties. In the following section, we define a metric to calculate the non-functional QoS equivalence degree for the services that are equivalent and almost equivalent.

3.3 Non-functional QoS equivalence degree

Services can be semantically equivalent, almost equivalent, or having *Subsume* matching relations. These equivalence are based on the functional aspect of services. Services can offer the same functionalities but with different non-functional QoS properties. We will define a metric that measures the non-functional QoS degree of equivalence. This metric allows to assign a normalised degree that measures the degree of non-functional QoS similarities between two equivalent, almost equivalent, or *Subsume* matching services. These degrees are used to choose between diverse services providing different non-functional QoS properties, but offering similar functionalities.

The non-functional QoS of an operation is defined as follows:

Definition 26: Non-functional QoS properties.

Consider a finite set of grammatical alphabet Σ, ontologies *O*, concepts *N* belongings to these ontologies *O*, non-functional QoS properties *Np*, quantitative non-functional properties Np_{QN}, and qualitative non-functional properties Np_{QL}. Considering an operation op we define its non-functional QoS as follows:

$$Np : \left\{ Np_{QL}^{*}, Np_{QN}^{*} \right\}$$

$$Np_{QL} = \left\{ np1_{QL}, np2_{QL}, \ldots npk_{QL} \right\},\ k = \left| NP_{QL} \right|$$

$$Np_{QN} = \left\{ np1_{QN}, np2_{QN}, \ldots npk_{QN} \right\},\ k = \left| NP_{QN} \right|$$

$$np_{QL} =< name, numericValue, operator >, name \in \Sigma^{*} \& numericValue \in \mathbb{R}$$

$$operator =< o, n >, o \in O, n \in N$$

operator specifies the order applied to *numericValue*. For {>, ≥} the greater the *numericValue* is, the best is the QoS property for the service runtime execution. For {<, ≤} the smaller the *numericValue* is, the best is the QoS property for the service runtime execution.

The non-functional equivalence degree $QoS_{Degree}(opi, opj)$ between two functional equivalent operations is evaluated upon their quantitative and qualitative properties similarities. Two functional equivalent operations offer the same functionality but not necessarily the same non-functional QoS properties. The $QoS_{Degree}(opi, opj)$ evaluates the degree of similarities of two operations opi and opj concerning their non-functional QoS properties. We suppose that:

$$\exists f : Np_{opi} \rightarrow Np_{opj} \ where\ \forall npk_{opj} \in Np_{opj}, \exists! npk_{opi} \in Np_{opi}, f\left(npk_{opi}\right) = npk_{opj}.$$

$\forall\ k \in \mathbb{N}$, npk_{opi} and npk_{opj} deals with the same non-functional QoS property. If npk_{opi} is a quantitative non-functional QoS we have npk_{opj} also a quantitative non-functional QoS and $name_{npk_{opi}} = name_{npk_{opj}}$. If npk_{opi} is a qualitative non-functional QoS we have npk_{opj} also a qualitative non-functional QoS and $name_{npk_{opi}} = name_{npk_{opj}}$.

Definition 27: $QoS_{Degree}(opi, opj)$

Considering two operations *opi* and *opj*, we define the degree of equivalence between the two operations $QoS_{Degree}(opi, opj)$ as a function that measures how close is *opj* from *opi* in terms of non-functional QoS. We consider the non-functional properties of *opi*, NP_{opi} and calculate as follows the degree of equivalence *opj* has upon these properties:

$$QoS_{Degree}(opi, opj) = \sum_{k=1}^{|Np_{opi}|} w_k * \deg\left(npk_{opi}, npk_{opj}\right)$$

where w_k is the assigned weight for a particular non-functional QoS property with the following conditions $\sum_{k=1}^{|Np_{opi}|}(w_k) = 1.$ The more w_k is closer to zero, the more important is the property *Npk*. This ponderation allows to decide when searching for equivalent services if certain non-functional QoS properties are more important than other for the required service replacement. deg(npk_{opi}, npk_{opj}) are normalised values between 0 and 1 corresponding to the equivalence degree between npk_{opi} and npk_{opj}. These values are calculated using the z-score or standardisation of the *npk* values for quantitative properties and semantic distance for qualitative properties.

We define deg(npk_{opi}, npk_{opj}) as follows:

- $\deg(npk_{opi}, npk_{opj}) = \deg(npk_{QN_{opi}}, npk_{QN_{opj}})$ for the quantitative properties
- $\deg(npk_{opi}, npk_{opj}) = \deg(npk_{QL_{opi}}, npk_{QL_{opj}})$ for the qualitative ones.

We define next how we calculate these two degrees.

Definition 28: $\deg(npk_{QN_{opi}}, npk_{QN_{opj}})$

$$\deg\left(npk_{QN_{opi}}, npk_{QN_{opj}}\right) = \left|\eta\left(npk_{QN_{opi}}\right) - \eta\left(npk_{QN_{opj}}\right)\right|$$

We define $\eta(npk_{QN})$ as the normalisation of z-score value of npk_{QN} for quantitative non-functional QoS.

Definition 29: $\eta(np_{QN})$

Considering np_{QN} =<*name*, *numericValue*, *operator*> we define $\eta(np_{QN})$ as follows:

$$\begin{aligned} &\textit{if operator}_{np_{QN}} \textit{ is}' <': 0 \textit{ if } z\text{-}score\left(np_{QN}\right) < -2 \\ &\qquad 1 \textit{ if } z\text{-}score\left(np_{QN}\right) > 2 \\ &\qquad \left(z\text{-}score\left(np_{QN}\right)\right)/4 + 0.5 \textit{ if } 2 > z\text{-}score\left(np_{QN}\right) > -2 \end{aligned}$$

$$\begin{aligned} &\textit{if operator}_{np_{QN}} \textit{ is}' >': 1 \textit{ if } z\text{-}score\left(np_{QN}\right) < -2 \\ &\qquad 0 \textit{ if } z\text{-}score\left(np_{QN}\right) > 2 \\ &\qquad 0.5 - \left(z\text{-}score\left(np_{QN}\right)\right)/4 \textit{ if } 2 > z\text{-}score\left(np_{QN}\right) > -2 \end{aligned}$$

For < the *numericValue* is the best when it is the smallest. $\eta(np_{QN})$ is closer to zero for the smallest value of *numericValue* and closer to one for the bigger value of *numericValue*, and vice versa for >.

The z-score of a quantitative property np_{QN}, indicates how far and in what direction, the property deviates from its distribution's mean, expressed in units of its distribution's standard deviation. We use the z-score standardisation in order to provide a way of comparing all the different non-functional QoS by including consideration of their respective distributions.

Definition 30: *z-score*(np_{QN})

Considering the quantitative np_{QN}, its corresponding z-score is:

$$z\text{-}score\left(np_{QN}\right)=\left(numericValue_{npQN} - \mu\left(numericValue_{np_{QN}}\right)\right)\Big/\sigma\left(numericValue_{np_{QN}}\right)$$

where $\mu(numericValue_{np_{QN}})$ is the mean of the values of np_{QN}, and $\sigma(numericValue_{np_{QN}})$ is the standard deviation of np_{QN}.

In normal distribution we can distinguish that the 95% of z-score(np_{QN}) values are comprises between −2 and 2. Based on this, $\eta(np_{QN})$ calculates a value between 0 and 1 taking into account the nature of quantitative non-functional QoS properties. Indeed the $operator_{np_{QN}}$ indicates whether the properties are stronger with greater values, or with smaller values.

If for the quantitative non-functional QoS properties, we used z-score and normalisation to calculate the degree of similarities between two properties, for qualitative non-functional QoS we use the semantic distance to compare the concepts of the qualitative properties np_{QL}. The semantic distance returns a normalised value between 0 and 1.

Definition 31: $\deg(npk_{QL_{opi}}, npk_{QL_{opj}})$

Considering $npk_{QL_{opi}}$ the qualitative non-functional QoS of the operation. We seek to find the best equivalence for it from a set of equivalent operations. Considering $npk_{QL_{opj}} = < name, semantic >$ the qualitative non-functional QoS of the other operations. We define $\deg(npk_{QL_{opi}}, npk_{QL_{opj}})$ as follows:

$$\deg\left(npk_{QL_{opi}}, npk_{QL_{opj}}\right) = D_{sem}\left(n_{semantic_{npkQL_{opi}}}, n_{semantic_{npkQL_{opi}}}\right)$$

Example 9: Considering the three operations defined in Figure 1. Considering the *Printing* operation, it is almost equivalent to *Printer* and almost equivalent to *Impression*. We calculate the non-functional QoS degree of equivalence to determine which of *Printer* or *Impression* replace the best *Printing*.

First, we calculate the values that we need for our degree computing. We detail the computing for *nbpage*.

$$\mu(nbpage) = 56.66$$
$$\sigma(nbpage) = \sqrt{\left((60-56.66)^2 + (100-56.66)^2 + (10-56.66)^2\right) \div 3} = 36.84$$
$$z\text{-}score\left(nbpage_{printing}\right) = (60-56.66) \div 36.84 = 0.09$$
$$z\text{-}score\left(nbpage_{impression}\right) = (100-56.66) \div 36.84 = 1.176$$
$$z\text{-}score\left(nbpage_{printer}\right) = (10-56.66) \div 36.84 = -1.26$$
$$\eta\left(nbpage_{printing}\right) = 0.477$$
$$\eta\left(nbpage_{impression}\right) = 0.206$$
$$\eta\left(nbpage_{printer}\right) = 0.816$$

$$\eta\left(nbpage_{printing}\right) = 0.515$$
$$\eta\left(nbpage_{impression}\right) = 0.867$$
$$\eta\left(nbpage_{printer}\right) = 0.186$$

$$D_{sem}\left(access_{printing}, access_{printing}\right) = 0,\ M_{Cpt}('wifi', 'wifi') = Exact$$
$$D_{sem}\left(access_{printing}, access_{impression}\right) = 0.2\ M_{Cpt}('wireless', 'wifi') = PlugIn$$
$$D_{sem}\left(access_{printing}, access_{printer}\right) = 1\ M_{Cpt}('bluetooth', 'wifi') = Fail$$

The QoS_{Degree} of the three operations are:

$$\begin{aligned} QoS_{Degree}(Printing, Impression) = {} & w1 * \left(\left|\eta\left(nbpage_{printing}\right) - \eta\left(nbpage_{impression}\right)\right|\right) \\ & + w2 * \left(\left|\eta\left(price_{printing}\right) - \eta\left(price_{impression}\right)\right|\right) \\ & + w3 * \left(D_{sem}\left(access_{printing}, access_{impression}\right)\right) \end{aligned}$$

$$QoS_{Degree}(Printing, Impression) = w1 * 0.27 + w2 * 0.35 + w3 * 0.2$$

$$QoS_{Degree}(Printing, Printer) = w1 * 0.33 + w2 * 0.33 + w3 * 1$$

If we suppose the three non-functional QoS properties of the same importance $w1 + w2 + w3 = 1$, we obtain: QoS_{Degree}(*Printing*, *Impression*) = 0.27, and QoS_{Degree}(*Printing*, *Printer*) = 0.55. The *Impression* operation offers non-functional QoS that are closer to *Printing* than *Printer* if we assign the same weight to the three non-functional properties.

4 Semantic service substitution in pervasive environments

An application executing a service in pervasive environments would like to benefit from all the available services. Service substitution based on semantic interface matching and non-functional QoS properties is something the pervasive environment can provide to applications. We use the equivalence and almost equivalence relations to compare services together to know if one service can substitute another one. And we use the QoS degree equivalence to be sure that the services we provide to applications fit their needs. When a service appears in the environment, it can be functionally equivalent to another service being executed by an application and with better QoS parameters. The environment will spontaneously substitute the service of the application with this new service. On the other hand, when a service disappear, the environment will look for equivalent or almost equivalent services with QoS properties similar to the vanishing services and redirect the application calls to this new service. These two actions of spontaneously substituting services to applications allow these latter to execute properly despite the environment dynamicity.

4.1 Service appearance

Considering a set S of finite services in the environment, we denote *si* the service that appears. As a first step, the pervasive environment searches for functionally equivalent or almost equivalent services interfaces in the environment. Indeed, these services are services that provide the same functionality – the same functional interfaces – as the service *si*, and can be replaced in the application clients execution by the service *si*.

We consider the new service *si*. We suppose that the service *si* is equivalent or almost equivalent to other services in the environment:

$$\exists\, sj \in S, \left(\equiv_{sem} (si, sj) = true\right) \vee \left(\rhd_{sem} (si, sj) = true\right)$$

The spontaneous service *si* substitution succeeds if *si* can replace *sj* for the application execution and that by providing better non-functional QoS properties than *sj* for the applications. By checking the profile of applications, the pervasive environment knows the values and the priorities (*wi*) that the applications would like to assign to the non-functional QoS properties. The environment can simulate a service *sk*, with these values, and calculates the QoS_{degree} using the *wi* specified by the applications. If no *wi* are assigned, the pervasive environment applies the following values: $\sum_{i\in\mathbb{N}} wi = 1$. The service substitution succeeds if:

$$QoS_{degree}(si, sk) < QoS_{degree}(sj, sk)$$

Which means that the new service *si* is closer to *sk* than *sj* is to *sk* in terms of non-functional QoS properties, *sk* reflecting the applications needs and preferences for the non-functional QoS properties of the service they execute.

Example 10: Considering the three operations defined in Figure 1.

The *Printing* service is a new service appearing in the environment and is semantic almost equivalent to the *Impression* service. The environment considers applications using the *Impression* service, and verifies which non-functional QoS properties are the required by the applications. For example, if the price is important, the w_{price} would be much more important than the w_{access} and w_{nbPage}, and the new *Printing* service fits better for the application. The environment simulates a new service by assigning it the adequate values of the non-functional QoS properties required by applications. As an example, we can give the following application required non-functional QoS properties depicted under service *sk*:

$$NP_{QL} = \{< access, “wireless" >\}$$
$$NP_{QN} = \{< nbPage, 50, '>' >, < price, 12, '<' >\}$$

And $w_{price} = 0.6$, $w_{access} = 0.2$, $w_{nbPage} = 0.2$.

First, we calculate the values that we need for our degree calculations:

$$\text{The mean for nbpage property}: \mu(nbpage) = 55$$
$$\text{The standard deviation for nbpage property}: \sigma(nbpage) = 32$$
$$\text{The normalised z-score values are}: \eta\left(nbpage_{printing}\right) = 0.46$$
$$\eta\left(nbpage_{impression}\right) = 0.149$$
$$\eta\left(nbpage_{printer}\right) = 0.85$$
$$\eta\left(nbpage_{sk}\right) = 0.54$$

$$\text{The mean for price property}: \mu(price) = 11$$
$$\text{The standard deviation for price property}: \sigma(price) = 6,4$$
$$\text{The normalised z-score values are}: \eta\left(price_{printing}\right) = 0.46$$
$$\eta\left(price_{impression}\right) = 0.85$$
$$\eta\left(price_{printer}\right) = 0.15$$
$$\eta\left(price_{sk}\right) = 0.539$$

The semantic distance for the non-functional properties are:

$$D_{sem}\left(access_{printing}, access_{sk}\right) = 0.8, \; M_{Cpt}('wifi', 'wireless') = Subsume$$
$$D_{sem}\left(access_{printing}, access_{sk}\right) = 0 \; M_{Cpt}('wireless', 'wireless') = PlugIn$$
$$D_{sem}\left(access_{printer}, access_{printer}\right) = 1 \; M_{Cpt}('bluetooth', 'wireless') = Fail$$

Using these values we calculate:

$$QoS_{degree}(Printing, sk) = 0.6 * 0.08 + 0.2 * 0.8 + 0.2 * 0.078 = 0.22$$
$$QoS_{degree}(Impression, sk) = 0.6 * 0.391 + 0.2 * 0 + 0.2 * 0.311 = 0.29$$

We have $QoS_{degree}(Printing, sk) < QoS_{degree}(Impression, sk)$, which means that the new printing service fits better the application requirements.

4.2 Service disappearance

Another major issue requiring service substitution is the disappearance of services form the environment. If a service disappears, the service registry of the environment is notified. This one asks the environment to come back with all the services that are equivalent or almost equivalent to this service. If many services are found, the environment creates sets of services. A set for the services equivalent and another one for the almost equivalence. The equivalence is considered better than the almost equivalence, as services can be interchanged in an equivalence relation (symmetric relation).

We denote *si* the service that disappears and for this service the environment finds the equivalent or almost equivalent services:

$$\exists\, sj \in S, \left(\equiv_{sem} (sj, si) = true\right) \vee \left(\rhd_{sem} (sj, si) = true\right)$$

We define the following:

$$S_{\equiv} : set\ of\ sj, \left(\equiv_{sem} (sj, si) = true\right)$$
$$S_{\rhd} : set\ of\ sj, \left(\rhd_{sem} (sj, si) = true\right)$$

In every set, services are ordered following the QoS_{degree} function that returns for every equivalent services with the service that disappeared their degree of equivalence concerning the non-functional QoS properties related to the service that the environment would like to replace.

By checking the values on the non-functional QoS properties for each service of every set, the environment calculates the $QoSd_{egree}(sj, si)$, $\forall\, sj \in S_*$, of each service of a set with the service *si*. If no ponderation is given by the applications upon the priority of the properties the environment employs the same value for $wi : \sum_{i\in\mathbb{N}} wi = 1$. The services within each set are ordered from the best one [service *sj* that minimises $QoS_{degree}(sj, si)$) to the worst one [service *sk* that maximise $QoS_{degree}(sj, si)$):

$$T_{\equiv} : set\ of\ ordered\ sj, \left(QoS_{degree}(sj, si) < QoS_{degree}\left(s_{j+1}, si\right), j \in \left[1 \ldots |S_{\equiv}| - 1\right]\right)$$
$$T_{\rhd} : set\ of\ ordered\ sj, \left(QoS_{degree}(sj, si) < QoS_{degree}\left(s_{j+1}, si\right), j \in \left[1 \ldots |S_{\rhd}| - 1\right]\right)$$

When a service *si* disappears, the environment chooses the best replacement for the service *si* by beginning from the most suitable set with the most suitable non-functional QoS properties.

Example 11: Returning to our example of the *Printing*, *Impression*, and *Printer* services (c.f., Figure 1).

If we search to replace the *Printing* service because of a sudden disappearance and need to choose between the *Impression* or the *Printer* services, the calculated QoS_{degree} between these services are different depending on the values assigned to *wi*.

$$\begin{aligned} QoS_{Degree}(Printing, Impression) = {} & w1 * \left(\left|\eta\left(nbpage_{printing}\right) - \eta\left(nbpage_{impression}\right)\right|\right) \\ & + w2 * \left(\left|\eta\left(price_{printing}\right) - \eta\left(price_{impression}\right)\right|\right) \\ & + w3 * \left(D_{sem}\left(access_{printing}, access_{impression}\right)\right) \end{aligned}$$

$$QoS_{Degree}(Printing, Impression) = w1 * 0.27 + w2 * 0.35 + w3 * 0.2$$

$$\begin{aligned} QoS_{Degree}(Printing, Printer) = {} & w1 * \left(\left|\eta\left(nbpage_{printing}\right) - \eta\left(nbpage_{impression}\right)\right|\right) \\ & + w2 * \left(\left|\eta\left(price_{printing}\right) - \eta\left(price_{printer}\right)\right|\right) \\ & + w3 * \left(D_{sem}\left(access_{printing}, access_{printer}\right)\right) \end{aligned}$$

$$QoS_{Degree}(Printing, Printer) = w1 * 0.33 + w2 * 0.33 + w3 * 1$$

If the service *Printing* is no longer available, the environment finds the services *Impression* and *Printer* as almost equivalent to *Printing*. For their non-functional properties, it is clear that if the environment assigns the same value to the three *wi*, the *Impression* service would have a closer degree to *Printing*. Nevertheless, if the application using *Printing* gives more importance to the price of the printing service, the environment will assign to *w*2 a greater importance, and we can notice the *Printer* service has a closer degree to *Printing* than the *Impression* service.

It can occurs that no equivalent or almost equivalent services are found, in that case the search may be refined over a set of operations. If the users and applications of the services that disappeared used a particular operation or set of operations, the search may be specified over these operations using the equivalence and almost equivalence service relations defined upon particular operations $(\equiv_{sem}^{Op}, \rhd_{sem}^{Op})$.

The spontaneous service *si* substitution over a predefined set of operations *Op* succeeds if:

$$\exists\, sj \in S, \left(\equiv_{sem}^{Op}(sj, si) = true\right) \vee \left(\rhd_{sem}^{Op}(sj, si) = ture\right)$$

Example 12: Considering the three services interfaces and their semantic descriptions in Figure 2:

We have $(\rhd_{sem}^{\{op1ifc1, op2ifc1\}}(ifc1, ifc2) = true)$, which means that the services proposing the interface *ifc*1 with the operations $op1_{ifc1}$ and $op2_{ifc1}$ can replace the operations $op1_{ifc2}$ and $op3_{ifc2}$ of service *ifc*2.

As for the service as a whole, the environment requires to create the sets of equivalent and almost equivalent services over the predefined set of operations. It also orders the services within these sets depending on the non-functional QoS properties of the concerned operations and not the non-functional QoS properties of all the service.

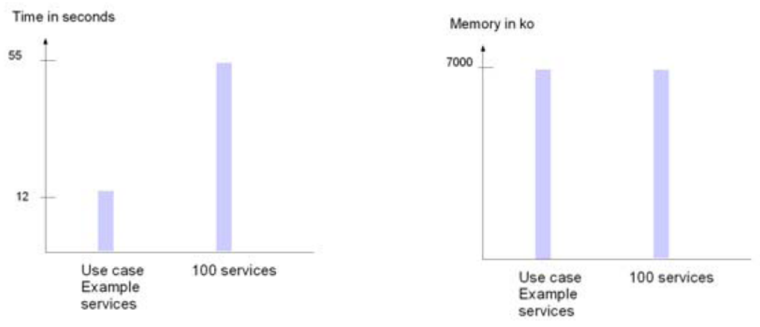
If no services are found, the environment may consider the services that are *Subsume* matching with the service that disappeared. If this replacement can fail to provide the required functionality as a *Subsume* matching between services does not guarantee that the new service can provide all what the other service provided, it can allows the environment to provide something to the applications even if not exactly what is required, while awaiting the appearance of the desired services. The environment proposes these services to the applications, specifying that the services they seek are no longer available.

In case of complete failure of finding an appropriate service, the service registry of the environment redirects all the calls to the functional interface of the disappearing service to a proxy. Once a service registers a functional interface responding to the applications needs, the calls of the proxy can be redirected to this new service.

5 Evaluation of the semantic service substitution

We implemented, as a proof of concept, all the major functionalities of the service substitution under an OSGi service platform implementation, the Apache Felix. The service semantic matching is done using online reasoner OWL-S ontologies (The OWL Services Coalition, 2005) and the matching relations of Paolucci et al. (2002). The non-functional QoS properties are for now defined in the service description and we do not yet consider the dynamic changes affecting these properties while service execution. For the evaluations we developed a use case composed of 100 OSGi services in a small environment deployed on three laptops (Dell Latitude D410, 1.73 GHz, and 0.99 Go of memory).

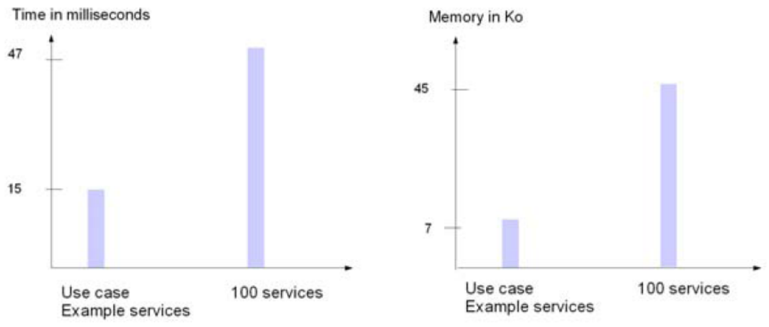
Figure 5 Time execution for semantic service matching (see online version for colours)



The semantic matching is quite heavy (c.f., Figure 5). The OWL-S API takes about 12 seconds to compare and matches 8 services owl-s descriptions (*MyStudio*) and 55 seconds for about 100 services. The pellet matching engine that reads all the OWL-S files by adding them to the reasoner and extracts the inputs, outputs and concepts fields is much slower and much more memory consumer than as simple syntactic matching based for example on introspection methods provided by the Java language. We conclude that the semantic matching using online semantic reasoning is a very heavy process. We can improve the matching time and memory consuming by employing techniques as in PERSE (Mokhtar, 2007) that propose efficient semantic service matching using encoding classified ontologies.

Figure 6 gives the time execution and memory consumption for quantitative non-functional properties QoS_{degree} function computing. We suppose that each service has one quantitative non-functional property. When a service leaves the environment, the time to adapt to these changes is the time required to compute and sort the QoS degree of available services publishing the same interfaces (47 milliseconds for 100 services). When a service appears in the environment, the environment computes the QoS degree of these services to find if it better suits the applications using equivalent services. If so, the service registry will propose to applications the new service and the adaptation would be done in no time for the application, as it is showed Figure 6.

Figure 6 Time and memory consumption for QoS degree computing (see online version for colours)



6 Conclusions

Service substitution is used in runtime reconfiguration in SOA systems in order to tolerate runtime variations and ensure continuity in service provisioning for the users. Providing functionally equivalent services to the applications with better quality of services when services appear and disappear is a challenging problem as services are provided with different technologies and different characteristics. If many middleware proposed to semantically compare services and to adapt them to the application execution, few formalised and defined the service relations and especially the non-functional QoS properties degree metrics between services. We proposed a metric to compare services, based on semantic interface matching and a metric for computing the non-functional QoS property similarities between services. We implemented a prototype under Java OSGi framework as a proof of concept and evaluated the efficiency of our proposal.

One of the aspects that are not yet tackled by our middleware prototype is the state of a service (Preuveneers and Berbers, 2008) that disappears while executing. If a service disappears while executing an application needs, to replace it in a transparent way, the environment needs not only to find equivalent services in terms of functional and non-functional QoS properties but to know from which state to start the execution of the new service, so that the application does not loose what has been already executed by the previous service. Mechanisms of logging and checkpoints need to be introduced at the service execution time level to save the state of a service at runtime. These mechanisms allow the environment to keep a trace over the state of services and to know when they disappear at which state of execution they were. Another important issue would be to test our prototype in large pervasive environments, such as university campus, were thousands of services may meet and where a real end user experience could be tested to evaluate the interest of our spontaneous service substitution approach vis à vis to users. Our approach would surely have problem to scale to these service numbers and a more smart selection, based not only on semantic ontologies but also on user profiles, would be appropriate to choose a subset of services to substitute.

References

Aït-Bachir, A. and Fauvet, M-C. (2009) 'Diagnosing and measuring incompatibilities between pairs of services', *DEXA '09: Proceedings of the 20th International Conference on Database and Expert Systems Applications*, pp.229–243, Linz, Austria.

Bittner, T., Donnelly, M. and Winter, S. (2005) 'Ontology and semantic interoperability', in Prosperi, D. and Zlatanova, S. (Eds.): *Large-Scale 3D Data Integration: Challenges and Opportunities*, pp.139–160, CRC Press, Tailor & Francis.

Erl, T. (2005) *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*, Prentice Hall PTR, Upper Saddle River, NJ, USA.

Floch, J. (Ed.) (2006) *Theory of Adaptation*, Delivrable D2.2, Mobility and ADaptation enAbling Middleware (MADAM).

Fredj, M., Georgantas, N. and Issarny, V. (2008) 'Dynamic service substitution in service-oriented architectures', *SCC '08: Proceedings of the IEEE Conference on Services Computing*, pp.101–104, Honolulu, Hawaii, USA.

Kokash, N. (2006) 'A comparison of web service interface similarity measures', *Proceeding of the 2006 Conference on STAIRS 2006*, pp.230–231, Amsterdam, The Netherlands.

Mokhtar, S.B. (2007) *Semantic Middleware for Service-Oriented Pervasive Computing*, PHD thesis, University of Paris 6.

Paolucci, M., Kawamura, T., Payne, T.R. and Sycara, K. (2002) 'Semantic matching of web services capabilities', *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pp.333–347, London, UK.

Preuveneers, D. and Berbers, Y. (2008) 'Pervasive services on the move: smart service diffusion on the OSGi framework', *UIC '08: Proceedings of the 5th International Conference on Ubiquitous Intelligence and Computing*, pp.46–60, Oslo, Norway.

Santhanam, G.R., Basu, S. and Honavar, V. (2009) 'Web service substitution based on preferences over non-functional attributes', *SCC '09: Proceedings of the IEEE International Conference on Services Computing*, pp.210–217, Bangalore, India.

Satyanarayanan, M. (2001) 'Pervasive computing: vision and challenges', *IEEE Personal Communication*, August, Vol. 8, No. 4, pp.10–17.

The OWL Services Coalition (2003) *OWL-S: Semantic Markup for Web Services*, White paper, OWL Services Coalition.

Weiser, M. (1991) 'The computer for the 21st century', *Scientific American*, Vol. 265, No. 3, pp.94–104.