Journal of Theoretical and Applied Information Technology
15th November 2021. Vol.99. No 21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TOWARDS A CONTEXT-DRIVEN TRANSACTIONAL SERVICE SELECTION MECHANISM IN UBIQUITOUS AND PERVASIVE ENVIRONMENTS

[1]ETTAZI WIDAD, [2]RIANE DRISS, [3]NASSAR MAHMOUD
IMS Team, ADMIR Laboratory, ENSIAS, Mohammed V University of Rabat, Morocco
[1]widad.ettazi@um5s.net.ma, [2]riane.driss@gmail.com, [3]nassar@ensias.ma

ABSTRACT

Most prominent techniques and selection algorithms only support QoS settings for application services. However, the software, hardware, and network infrastructures underlying services and users application have a significant impact on the validation of transactional services. Additionally, users may have varying transactional requirements throughout the lifecycle of a service composition. Therefore, selection algorithms must take into account the context requirements and users transactional needs when selecting services. This has led us to explore the trail of a service selection mechanism based on a service description enriched by functional and transactional requirements, and context information. We propose a context-driven selection of transactional services by introducing a new selection algorithm CT2S based on a semantic matching mechanism. More precisely, we are interested in studying the response time of our CT2S algorithm vis-a-vis the rapidity requirements in pervasive environments.

Keywords: *Context-Awareness, Transactional Service, Semantic Matching, Selection Mechanism, Service Discovery.*

1. INTRODUCTION

With the advent of mobile computing, the dynamic nature of application execution context induces multiple transactional needs during the lifecycle of a transaction. The transfer of funds between bank accounts and the booking of airline seats and hotel rooms are the classic examples of transactional applications. The spectrum of this kind of applications has quickly expanded to include CAO, e-commerce, workflow management, etc.

The proliferation of these paradigms has led to an evolving subject area known as pervasive computing. The latter is an intuitive evolution of computing paradigms driven by the wide adoption of mobile devices and wireless networks. Systems are now expected to adjust to user's requirements and customize their services to user's needs. Nevertheless, supporting user's tasks from a functional point of view is not enough to gain his satisfaction. In pervasive and ubiquitous environments, transactions must be able to adjust to systems that are not necessarily in a perfect environment, for example, that don't require a lock of their resources and do not care if transactions run for short periods of time or longer periods. These systems will operate in a flexible, dynamic environment, but less reliable and that presents contextual requirements (i.e., requirements and preferences expressed or implied by the user, connectivity, bandwidth, etc.) that hinder the transactions execution [1], [2] and [3].

Service selection is an essential condition for the composition of services in so-called pervasive environments. Despite the multitude of research works on selection algorithms, to our knowledge, there are no algorithms dealing with the major issues imposed by pervasive environments on the selection of transactional services. In addition, the context-driven selection of services exhibiting transactional properties raises several challenges. CATS (Context-Aware Transactional Service) composition in pervasive environments generally involves dynamic execution contexts, service unavailability and varying user requirements [4]. Thus, the selection techniques of such services must be designed to be proactive. Indeed, context-aware computing envisages satisfying user tasks on the fly, therefore, the time available for the selection and composition of services is limited compared to the complexity of requests processing. Existing algorithms developed for service selection need to be reviewed and possibly revisited. Service selection according to context requirements is already discussed in the literature [5]. However, prospective researches are primordial to fit pervasive environments particularities (e.g.

resource limitations, device mobility, wireless network connectivity), the characteristics of transaction processing (e.g. timeout, response time) and the transactional needs of users. Furthermore most of the existing service selection algorithms are developed, solely, based on the context information delivered by service providers. Additionally, at run time, the current context may fluctuate with respect to the provided context due to changes that may occur in the pervasive environment (e.g., user mobility, service unavailability). To deal with this issue, selection algorithms must consider the context captured at run-time. This requires monitoring the context of all candidate services just before implementing the services, which is difficult to achieve considering the large number of services to be analyzed in order to accomplish the user task. Withal, the context-aware composition of services exhibiting transactional properties poses several challenges. A major challenge is the transactional behavior of candidate services which is subject to perpetual changes while the composition is running. Selection strategies must also take into account the transactional properties of services by adjusting them in relation to the execution context. Great efforts have been concentrated on semantic research and context adaptation, particularly adaptation to the location and the used devices. In the recent years, we see the limitations of these approaches, especially the user overload due to false-positive. Indeed, users are offered several implementations for the same service, without having the necessary background to understand these implementations, which is detrimental to the transparency of these systems, notably for services that manifest transactional properties. The novelty of our approach is to offer the service that meets user's needs, without being forced to understand details about the implementation or the constraints of the used devices.

To deal with the aforementioned issues, a selection process is proposed in order to hide the complexity of the implementation services in a heterogeneous and dynamic environment, and consequently to achieve the promised transparency and the desired efficiency of pervasive and ubiquitous environments. We argue that better consideration of the user's transactional requirement can lead to a better understanding of actual service usage, which in turn can improve the accuracy of the selected services. In addition, contextual information plays a central role in this service selection process because it influences the choice of the best strategy to meet the transactional needs of users. The service selection mechanism is based on a new service description enriched by functional and transactional requirements, and context information. The concept of requirement is used to expose transactional services and to implement a user-centric view in a given context. We propose a context-driven selection of transactional services by introducing a new selection algorithm CT2S based on a semantic matching mechanism.

This article is organized as follows. The section II and section III will be devoted to review some basic concepts and related work. In section IV, we introduce the proposed service description by extending the OLW-S profile with transactional and contextual information. Section V details the proposed context-driven selection mechanism for transactional services. Section VI presents encouraging experimental results demonstrating our proposition. Results discussion is carried out in section VII. Finally, we conclude in the section VIII.

2. BASIC CONCEPTS

We present in this part some backgrounds related to context-awareness and transactional concepts.

2.1 Context-Awareness

Context-aware computing appeared since the 90s driven by the work of [6]. This term refers to systems capable of perceiving a set of conditions of use in order to adjust their behavior in terms of providing information and services. According to [7], the definitions ascribed to a context-aware system do not include all types of context-aware systems. Indeed, under these definitions, a system that simply collects the context in order to provide it to an application is not considered a context-aware system. Thus, the authors believe that "a system is context-aware if it uses context to provide relevant information and services to the user, where relevance depends on the task requested by the user".

2.2 Transactional Service

We use the term Transactional Service (TS) to indicate a sequence of activities performed by a user in order to carry out a specific task or fulfill a specific goal by means of a service-oriented platform. In a context-aware transactional service, the execution of operations and the context-awareness are combined. The resulting complexity of CATS requires them to be designed prior to being implemented. Disregarding the context-awareness aspect, during the design process of transactional services, results in systems with low accommodation and inappropriate behaviors.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3. LITERATURE REVIEW

In recent years, great research efforts have been carried out on the subject of service selection. Indeed, the relevance of a service selection mechanism depends on how its matching algorithm allows going beyond what is already provided by standard mechanisms such as UPnP, Jini, etc. This subject was largely treated according to a semantic vision. Different works such as [8], [9] and [10] concentrated their efforts on semantic matching between the service capabilities and the user request. These works served as a basis for other works such as the researches in the field of context-aware service selection [11], [12] and [13]. These authors take into consideration the service context and the user's current context when selecting the most appropriate service. In addition, other works such as [14] and [15], following a TQoS approach have proposed service discovery mechanisms according to transactional requirements that take into consideration the transactional profile that a service is able to satisfy during the discovery process. First researches for semantic service discovery focused on the matching between inputs and outputs to select the most relevant service according to a given request [9], [10]. These authors propose semantic matching mechanisms presented in [8] and [10] and hybrid matching mechanisms in [9] based on the capabilities signatures provided by services. These mechanisms include the identification of subsumption relations between the concepts describing the service's inputs and service's outputs [16]. The relations are similar to inheritance relations and allow binding specific concepts to general ones, thus exploring the hierarchies between concepts in a given ontology. Context-awareness is the basis for different service discovery approaches [11], [12] and [17]. These approaches are mostly based on semantic descriptions of services. The context-awareness is one of the essential characteristics of pervasive systems, therefore, these approaches are particularly relevant for pervasive and ubiquitous computing, since they must adapt their service offers to the environment and the execution context. Reference [11] proposed a context-oriented approach for web service selection. The authors consider that the user and the service have contextual requirements for them to work properly. A user may have requirements related to the service context that he's looking for (e.g., availability, location, etc.), as well as the context provided by the execution environment (e.g., wireless connection, etc.). In return, a service may request contextual information about the user (e.g., location, terminal capabilities, etc.) and the environment (e.g., network, etc.). Similar to previous approaches, Reference [12] developed a semantic and service-oriented middleware, called EASY for the discovery and composition of services in a pervasive environment. The contribution includes the EASY-L language based on OWL, for unambiguous semantic specification of functional and non-functional service properties, and EASY-M (EASY-Matching) which represents a set of compliance relations for services matching in terms of their functional and non-functional properties. Reference [17] proposed a service selection mechanism based on context matching, which takes into account the uncertainty of context information when classifying service variants. This mechanism is based on OWL-S service description enriched with contextual properties. Reference [18] introduced a model describing the geographic space of a system to differentiate a set of execution contexts. Its main purpose is to help and guide designers to characterize possible evolutions in systems mobility during its future execution. The work presented in [19] used model transformation to generate a semantic representation of the context. The services are described by extending the OWL-S profile with contextual conditions. None of the previously cited works combines the notion of context with the concept of transactional requirements, unlike [14] and [15] and who have noted the importance of exploiting the close relation between these two concepts in the service discovery process. Reference [14] developed a framework for reliable replacement of transactional services driven by QoS parameters. The framework takes into consideration the QoS parameters of the reselection service, the transactional risk and the compensation cost during the replacement process. Reference [15] proposed a TQoS approach for the selection of services according to their transactional requirements, QoS characteristics and user preferences. The selection is made based on user needs in terms of transactional requirements and QoS features.

We believe that these approaches are in fact complementary, and that such an evolution can only truly be achieved by a combination of these approaches. To our notice, only a service selection mechanism based on both context and the requirement of the user is able to answer questions such as "why a service is useful in a given context?" or "under what circumstances emerges the need of a service?". We are convinced that the context cannot be reduced to simple input or output parameters. Not only does it influence the execution of the service, but it characterizes the service itself

and the transactional properties displayed by the service. According to our analysis, none of the cited works offers a mechanism for selecting services that actually combine and harness context and transactional profile. This selection mechanism is essential in user-centric pervasive and ubiquitous environments, which must be characterized by their adaptability to the context and their understanding of users' needs.

4. CATS OWL-SRC: ENRICHED SEMANTIC SERVICE DESCRIPTION FOR TRANSACTIONAL SERVICE DISCOVERY

In this section, we introduce the proposed service description from transactional and contextual perspectives.

4.1 Transactional Service Description

We consider that transactional services are running in a given context, but also that this type of service is supposed to satisfy transactional requirements of users. However, the service description used commonly does not necessarily represent both aspects. Thus, we propose to enrich the service description by introducing a semantic description of transactional services that merges both user's functional and transactional requirements which the services are supposed to satisfy, and the context in which these requirements are expressed and that can influence the execution of this type of service. In order to provide a semantic description, a rich service description language is necessary.

OWL-S [20] describes services in three interdependent sub-ontologies. The service profile exposes the service interface for service discovery. The process model describes the composition of the service, while the service grounding indicates how to invoke a service. The proposed extension essentially concerns the service profile. A service is no longer just a set of operations. A service must also correspond to a requirement (i.e., functional and transactional) that emerges in a given context (refer to figure 1).

From a requirement perspective, a user expresses a particular requirement when invoking a service. The service is offered to meet this requirement. Therefore, a requirement becomes central when defining the service. We propose to enrich the service description with the associated requirements. We extend the service profile in OWL-S description by the requirement that a transactional service can meet.

This extension is achieved by integrating a new "Requirement" parameter that characterizes the service profile elements. It is formulated according to a specific model, in which a "Requirement" is represented by a function, a resource and a transaction behavior. Figure 2 illustrates this extension. A service is associated with the "book ski session" requirement, which is described according to the "Function-Resource-Transaction behavior" model, using the extended OWL-S elements.

4.2 Context Description

Contextual information helps to fully understand user requirements. We consider that user requirements emerge in a particular context that gives them meaning. Conversely, context description provides valuable insight when it is associated with a requirement. Thus, we propose to enrich the OWL-S description with contextual information that characterizes the transactional service

On the one hand, we describe the context with reference to the conditions under which it is most appropriate to invoke the service. This description represents the context in which the requirement is expressed, and the service is intended to be entirely useful to the user. On the other hand, we describe the context in which a service can be executed. It refers to the conditions under which a service is running by the service provider. These two context descriptions represent the environment descriptor associated with a transactional service.

.

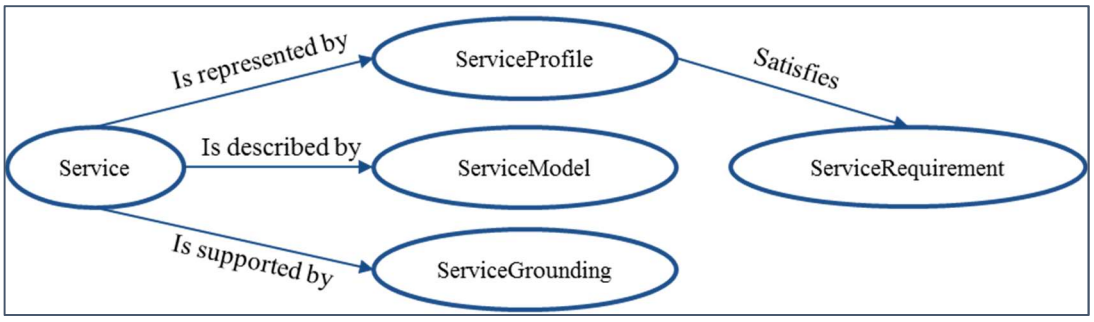

Figure 1: Service Requirement Description in OWL-S

```
<service: Service redf: ID="Ski_Lessons_Service">...</service:Service>
<profile:Profile rdf: ID="Ski_Lessons_Service">
...
  <profile:serviceName xml:lang="en"> ContextSkiLessonsService</profile:serviceName>
  <profile:textDescription xml:lang="en">
    This service provides a facility to book ski sessions, snowboard sessions
  </profile:textDescription

  <eprofile:context rdf:resource="http://193.55.96.54/iSOA/ExtensionOWL-S/
 ContextDescription.xml#contextdescriptor1"/>
    ...
  <etsprofile:TransactionalService rdf="http://193.55.96.54/iSOA/ExtensionOWL-S/
    TransactionalService.owl#transactionalservice">

    <etsprofile:Function rdf="http://193.55.96.54/iSOA/ExtensionOWL-S/
    TransactionalService.owl#transactionalservice.function.book"> Book </etsprofile:Function>

    <etsprofile:Resource rdf="http://193.55.96.54/iSOA/ExtensionOWL-S/
    TransactionalService.owl#transactionalservice.resource.object.accommodation">
        Ski Session
    </etsprofile:Resource>

    <etsprofile:TransactionBehavior rdf="http://193.55.96.54/iSOA/ExtensionOWL-S/
    TransactionalService.owl#transactionalservice.transactionbehavior.commit">
        Commit
    </etsprofile:TransactionBehavior>

    <etsprofile:TransactionBehavior rdf="http://193.55.96.54/iSOA/ExtensionOWL-S/
    TransactionalService.owl#transactionalservice.transactionbehavior.abort">
        Cancel
    </etsprofile:TransactionBehavior>

    <etsprofile:TransactionBehavior rdf="http://193.55.96.54/iSOA/ExtensionOWL-S/
    TransactionalService.owl#transactionalservice.transactionbehavior.compensate">
        Rollback
    </etsprofile:TransactionBehavior>

  </etsprofile:TransactionalService>
  .....
```

Figure 2: Transactional Service Description by Extending the Service Profile in OWL-S

Figure 3 describes the service requirement metamodel in a given context.

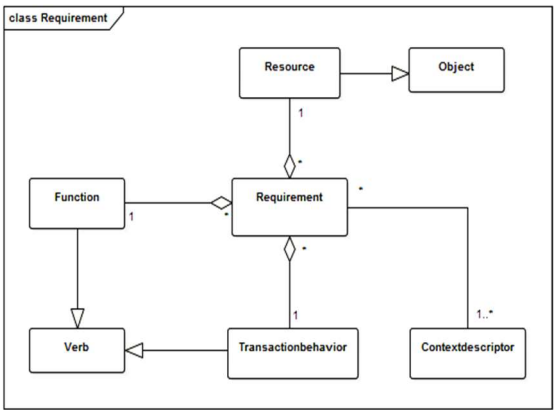

Figure 3: Service Requirement Metamodel in a Particular Context

As shown in figure 4, the environment descriptor is stored in an external file.

Based on the study proposed by [21], we enrich the service profile with the *contextdescriptor* attribute, which represents a URL pointing to the context description file. Such separation is necessary since the context information is dynamic and cannot be stored statically in OWL-S description. Thus, the service provider indicates that the service is suitable for adult users (line 12-19) with skill levels 4 to 6 (line 20-24), who are developing their skiing or snowboarding skills and exploring the mountains in a location near Ifrane, Morocco (line 30-39). Figure 4 exhibits the context description file, referenced by the extended service profile OWL-SRC shown in figure 2 (line 10). By referring to this context description in service profile, the service provider shows that the "book ski session" requirement associated with this service is expressed in the context described by the file displayed in figure 4, considered as a prerequisite for the execution of this service.

```
<ctx: context xmlns:ctx= "http://193.55.96.54/iSOA/ContextSchema"
        ...
        xsi:schemaLocation="http://193.55.96.54/iSOA/ContextSchema.xsd">
      <ctx:ContextDescriptorSet>
        <ctx:ContextDescriptor rdf: ID="contextdescriptor1">

                  <ctx:ContextParameter>
                    <ctx:hasGroup
resource="http://193.55.96.54/iSOA/Context.owl#concept.ContextGroup.user"/>
                    <ctx:hasDimension
resource="http://193.55.96.54/iSOA/Context.owl#concept.ContextDimension.userprofile"/>
                        ...
                    <ctx:ContextValueSet>
                      <ctx:ContextValue>
                        <ctx:hasDimension
resource="http://193.55.96.54/iSOA/Context.owl#concept.ContextDimension.profile.age"/>
                                ...
                        <ctx:Value> [18-60]  </ctx:Value>
                      <ctx:hasDimension
resource="http://193.55.96.54/iSOA/Context.owl#concept.ContextDimension.profile.abilitylevel"/>
                                ...
                        <ctx:Value> [4-6]  </ctx:Value>
                      </ctx:ContextValue>
                    </ctx:ContextValueSet>
                  </ctx:ContextParameter>

                  <ctx:ContextParameter>
                    <ctx:hasGroup
resource="http://193.55.96.54/iSOA/Context.owl#concept.ContextGroup.Environment"/>
                    <ctx:hasDimension
resource="http://193.55.96.54/iSOA/Context.owl#concept.ContextDimension.location"/>
                        ...
                    <ctx:ContextValueSet>
                      <ctx:ContextValue>
                        <ctx:hasDimension
resource="http://193.55.96.54/iSOA/Context.owl#concept.ContextDimension.location.city"/>
                                ...
                        <ctx:Value> IFRANE, MAROC  </ctx:Value>
                      </ctx:ContextValue>
                    </ctx:ContextValueSet>
                  </ctx:ContextParameter>
              </ctx:ContextDescriptor>
      <ctx:ContextDescriptor rdf: ID="contextdescriptor2"> ...  </ctx:ContextDescriptor>
    </ctx:ContextDescriptorSet>
...
```

Figure 4: Context Description File Referenced by Service Profile OWL-SRC

5. CONTEXT-DRIVEN TRANSACTIONAL SERVICES SELECTION MECHANISM

In this section, we introduce a new service selection mechanism based on a service description enriched by functional and transactional requirements and context information. The concept of requirement is used to expose transactional services and to implement a vision focused on user requirements in a given context.

5.1 CT2S Overview

We propose a context-driven transactional services selection based on a semantic matching algorithm. This matching is performed in a two-step process illustrated in figure 5.

Journal of Theoretical and Applied Information Technology
15th November 2021. Vol.99. No 21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

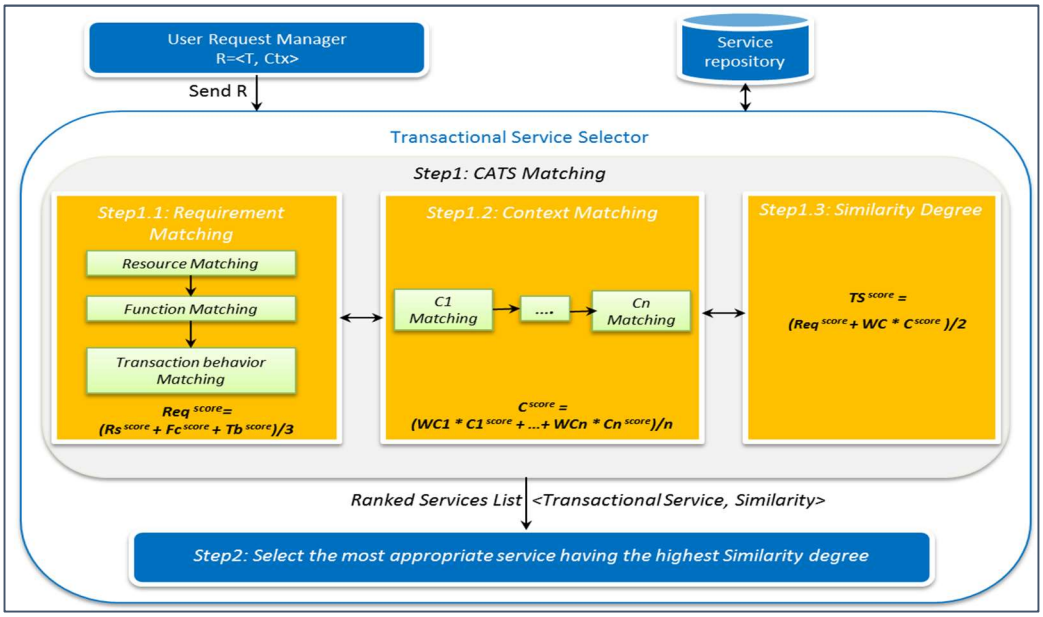

Figure 5: Context-Driven Transactional Service Selection Mechanism

First, the selection process proceeds to establish a correspondence between the user's requirement and the requirement that the transactional service allows to satisfy (step 1.1). Second, it matches the service environment descriptor with the current context of the user (step 1.2). Finally, it calculates the similarity degree between the user's request and the provided service, and adds the service and its calculated score in a list (step 1.3). Then, from the list of transactional services, the algorithm select the service with the highest similarity degree (step 2). The transactional service score represents the similarity degree of a provided service according to user's requirement and his current context.

Figure 6 shows the proposed selection algorithm. For each transactional service, the algorithm calculates the matching score TS^{score} between the user's request and the service (line 6-13). First, it calculates the requirement matching score Req^{score} between user requirement Req_U and service requirement Req_S (line 9). As we mentioned above, the requirement expresses the user's requirements that are to be satisfied by the system through his task. It consists of three main elements: Function (Fc), Resource (Rs) and Transaction behavior (Tb). The Function element exposes the action allowing the task to be fulfilled.

The Resource element represents either the object existing before the accomplishment of the task, or the result created by the action allowing the realization of the Function element. Thus, to define the similarity degree between the user's requirement Req_U and the service requirement Req_S, the algorithm calculates: (i) the similarity degree between the user's Resource elements and those of the service (respectively RS_U and RS_S), (ii) the similarity degree between the user's Function elements and those of the service (respectively Fc_U and Fc_S), (iii) the similarity degree between the user's Transaction behavior elements and those of the service (respectively Tb_U and Tb_S), (iv) the

Algorithm : Transactional Service Selection

1: **Procedure** SERVICE SELECTION (C_U, Req_U, TS, ontoFc, ontoRs, ontoTb, ontoC)
2: $TS_{ranked} = \emptyset$
3: $Req^{score} = 0$
4: $C^{score} = 0$
5: $TS^{score} = 0$
6: **for** $\forall si \in TS$ **do**
7: $C_{si} = getContext(si)$
8: $Req_{si} = getRequirement(si)$
9: $Req^{score} = RequirementMatching(Req_U, Req_{si}, ontoFc, ontoRs, ontoTb)$
10: **if** $Req^{score} > k$ **then**
11: $C^{score} = ContextMatching(C_U, C_{si}, ontoC)$
12: **if** $C^{score} > l$ **then**
13: $TS^{score} = (Req^{score} + C^{score})/2$
14: $TS_{ranked} = TS_{ranked} \cup \{<si, TS^{score}\}$
15: **end if**
16: **end if**
17: **end for**
18: $s_{selected} = getbestTransactionalService(TS_{ranked})$
19: **return** $s_{selected}$
20: **end procedure**

Figure 6: Context-Driven Transactional Service Selection Algorithm CT2S

Requirement Matching Score representing the sum of the respective matching scores of Function, Resource and Transaction behavior elements. Once the requirement has been matched, the algorithm proceeds to context matching, in which C^{score} is calculated based on the matching between the user's current context and the context descriptors established by service description (line 11). The two scores (i.e., Req^{score} and C^{score}) are then used to calculate the final service score TS^{score} (line 13). These two correspondences are detailed in the following sections.

5.2 Requirement Matching

Requirement Matching is based on the use of ontologies, semantic matching and similarity degree. With regard to the formulation of the requirement, the Requirement Matching is specially based on the correspondence of Function, Resource and Transaction behavior elements. Thereby, a Function element ontology, a Transaction behavior element ontology and a domain-specific ontology representing the possible Resource elements in a specific domain are utilized. The similarity degree is the distance calculated on the basis of the semantic link between two concepts in a given ontology. Hence, the Requirement Matching is calculated based on three relations, *ResourceMatch*, *FunctionMatch*, and *TransactionbehaviorMatch*, used to define the *RequirementMatch* relation between user's requirement $Req_U = <Fc_U, Rs_U, Tb_U>$ and service's requirement $Req_S = <Fc_S, Rs_S, Tb_S>$ as shown in the equation (1):

$$RequirementMatch(Req_U, Req_S) = \begin{cases} \forall\, Fc_U, \exists\, Fc_S\ FunctionMatch(Fc_U, Fc_S) \\ And \\ \forall\, Rs_U, \exists\, Rs_S\ ResourceMatch(Rs_U, Rs_S) \\ And \\ \forall\, Tb_U, \exists\, Tb_S\ TransactionBehaviorMatch(Tb_U, Tb_S) \end{cases} \quad (1)$$

5.2.1 Resource matchning

The *ResourceMatch* relation compares the concepts defined in a domain-specific ontology, depending on the resource required by the user Rs_U and the resource provided by the service Rs_S. The evaluation of the correspondence between a required resource and a provided resource is generally based on a hierarchy of subsumption used to determine which of the provided concepts corresponds to a required concept. In order to achieve such a correspondence, our algorithm is based on the semantic matching algorithm proposed by [8] using the following four levels:

- ✓ Exact: the required concept is equivalent to the provided concept.
- ✓ Plug-In: the required concept is included in the provided concept.
- ✓ Subsume: the required concept includes the provided concept.
- ✓ Fail: there is no subsumption between the two concepts.

Thus, the *Resource Matching* score is calculated on the basis of the *ResourceMatching* function which takes as input the domain-specific ontology, the user's Resource element and the service's Resource element. The result of this function represents the similarity degree between these two elements, calculated according to the distance between them in the domain-specific ontology. The Resource Matching score is calculated as shown in Table 1.

Table 1: Resource Matching Score Calculation Grid.

Matching Relation	Distance	Score
Exact	0	1
Fail	-1	0
Plug-In/Subsume	d	1/(d+1)

5.2.2 Function matchning

The The *FunctionMatch* relation is based on the Function element ontology, which contains a set of domain-specific functions, their different meanings and relations. Each relation associates a function with more general functions or with more specific functions or with functions which have a common meaning:

- ✓ Exact: the required function is equivalent to the provided function.
- ✓ Synonym: the required function has a common meaning with the provided function.
- ✓ Hyponym: there is a relation of subordination between the required function and the provided function with a more general meaning.
- ✓ Hypernym: there is a relation of subordination between the required function and the provided function with a more specific meaning.
- ✓ Fail: there is no relation between the two functions.

These levels are based on the relation Property (P, C_1, C_2), where C_1 is a required concept, C_2 is a provided concept, and P is the relation property between C_1 and C_2. This relation is defined in the equation (2):

$$Property\,(P, C_1, C_2) = \forall\, C_1, C_2, \exists\, P: C_1.P\; =\; C_2 \qquad (2)$$

Thus, the *Function Matching* score is calculated on the basis of the *FunctionMatching* function which takes as input the Function element ontology, the user's Function element and the service's Function element. The result of this function represents the Function Matching score. This score is calculated based on the relation between the user's Function element and the service's Function element. First, this function determines the property relation between these two elements. Then, the Function Matching score is calculated as shown in Table 2.

Table 2: Function Matching Score Calculation Grid.

Matching Relation	Score
Exact	1
Synonym	0.9
Hyponym	0.7
Hypernym	0.5
Fail	0

5.2.3 Transactional behavior matching

In this part, we define the semantics specifying the transactional properties exposed by the services that form the ontology of the Transaction behavior element. Our semantic model is based on the description of the transactional service defined in [22]. In this description, a model specifying the transactional properties of a service is presented. This model is based on the classification of computational tasks presented in [23], which consider three types of transactional properties. An operation or by extension a service executing a task can be:

- ✓ Compensable: The results produced by the task can be canceled.
- ✓ Re-executable: the task ends successfully after a finite number of attempts.
- ✓ Pivot: The task is neither compensable nor re-executable.

These transactional properties allow four types of transactional services to be defined: re-executable, compensable, re-executable and compensable and pivot. The *TransactionbehaviorMatch* relation is based on the *Transactionbehavior* element ontology, which contains a predefined set of transactional behaviors, their different meanings and relations. Each relation associates a transactional behavior with more general transactional behaviors, more specific transactional behaviors, and transactional behaviors that have a common meaning. Thus, we propose to classify the transactional behavior correspondences according to the five levels defined in the previous section (cf. 2) Function Matching).

Thus, the *Transactionbehavior* Matching score is calculated on the basis of the *TransactionbehaviorMatching* function which takes as input the *Transactionbehavior* element ontology, the user's *Transactionbehavior* element and the service's *Transactionbehavior* element. The result of this function represents the *Transactionbehavior* Matching score. This score is calculated based on the relation between the user's *Transactionbehavior* element and the service's *Transactionbehavior* element. First, this function determines the property relation between these two elements. Then the *Transactionbehavior* Matching score is calculated as shown in Table 2.

5.3 Context matching

The environment descriptor for a user C_U or a service C_S represents a set of context descriptors which are associated each with a group of a given context. Each context descriptor aggregates a set of observable parameters. Each context parameter is described by a context dimension that characterizes the property we are observing, and a set of observed values. In order to define the *ContextDescriptorMatch* relation, we consider the *ContextParameterMatch* relation which allows matching individually the different context parameters representing the user's context descriptors (C_U = {c_j} j> 0) and the service's context descriptors (C_S = {c_i} i> 0). This relation is presented in the equation (3):

$$ContextDescriptorMatch\,(C_S, C_U) = \forall\, c_i \in C_S\, P, \exists\, c_j \in C_U : ContextDescriptorMatch\,(c_i, c_j) \qquad (3)$$

Context parameters matching proceeds as follows: for each c_i and c_j, (i) the context matching mechanism initiates a matching process between the service context group $c_i.context$ and the user context group $c_j.context$; if the matching score between the two context groups is greater than a setting threshold, then (ii) the context matching process matches the service context dimension $c_i.contextdimension$ with the user context dimension $c_j.contextdimension$; if the matching score between the two context dimensions is greater than a setting threshold, then (iii) it matches the service context parameter $c_i.context$ with the user context parameter $c_j.context$, and if the matching score between the two context parameters is greater than a setting threshold, it matches the different observed values one by one. A previous work has developed in detail the context model ontology used for this purpose [4]. Context observable parameters can be classified into several types. Hence, the context could be represented as a multidimensional space.

Journal of Theoretical and Applied Information Technology
15th November 2021. Vol.99. No 21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Context observed values are distinguished between numeric and non-numeric types. In order to accommodate this diversity and the multidimensional space representing the context, the *ContextParameterMatch* relation identifies the type of the context parameter value and triggers the appropriate comparison function accordingly. This relation evaluates whether the user's context parameter is similar to the context parameter of the service based on a specific operator (e.g., equal, not-equal, between, higher-than, lower-than). Let's assume that the context descriptor of a service displayed that the device bandwidth should be greater than 12500. Based on the user's current context, if the captured value of user's device is actually greater than 12500, we get an exact match. Thus, the *ContextMatching* score C^{score} (refer to figure 7 line 11) is calculated as the sum of the scores of each context parameter, represented as in the equation (4).

$$C^{score} = \sum_{i,j=1}^{n} w_{c_i} f(type, c_i, c_j) \quad (4)$$

6. IMPLEMENTATION AND EVALUATION

The following section analyzes the results of the experiments that were carried out in order to assess the semantic matching process guided by transactional requirement and context.

The service discovery process, which we present in this article, has been implemented in Java language. This implementation is organized around a number of Java interfaces. The *IPersistenceManager* interface acts as a facade between the *TransactionalServiceSelector* component and the repository of services and ontologies allowing access and loading of services descriptions and ontologies. It maintains service descriptions and ontologies by offering methods of writing, reading, adding, deleting, loading, etc. The selection algorithm is implemented through the *ITSMatcherFaçade* interface of C/FRT Matchmaker component which is responsible for the matchmaking according to the C/FRT model (Context/Function-Resource-Transaction behavior). This component communicates with the context management component through its *IContextManager* façade to initiate the matchmaking between the user's current context and context descriptors of the required service. Then, it selects the most appropriate service which meets the immediate user's requirement in his current context. The *TransactionalServiceSelector* component loads through the *IPersistenceManager* persistence facade all the semantic descriptions of available services that are listed in the services repository [4].

```
1:  Procedure ContextMatching (C_U, C_si, ontoC)
2:  C^score = 0
3:  for ∀ cj ∈ C_si do   /* pour chaque descripteur de contexte cj */
4:      cd^score = 0     /* score du descripteur de contexte */
5:    Gp_cj = getContextGroup (cj)
6:    Dim_cj = getContextDimension (cj)
7:    Param_cj = getContextParameter (cj)
8:    w = getContextParameterweight(Param_cj)
9:  for ∀ c_U ∈ C_U do
10:    c^score = 0
11:    Gp_cu = getContextGroup (c_U)
12:    Gp^score = ContextMatch(Gp_cu, Gp_cj, ontoC)
13:  if Gp^score > k then
14:    Dim_cu = getContextDimension (c_U)
15:    Dim^score = ContextMatch(Dim_cu, Dim_cj, ontoC)
16:  if Dim^score > l then
17:    Param_cu = getContextParameter (c_U)
18:    Param^score = ContextMatch(Param_cu, Param_cj, ontoC)
19:  if Param^score > m then
20:    Val_cj = getContextParameterValue(c_j)
21:    Val_cu = getContextParameterValue(c_u)
22:    Unit_cj = getContextParameterUnit (c_j)
23:    Unit_cu = getContextParameterUnit(c_u)
24:   if UnitMatch (Unit_cu , Unit_cj) then
25:    c^score = ContextParameterValueMatch (Val_cu,Val_cj)
26:    cd^score = (Gp^score + Dim^score + w* Param^score + c^score ) / 4
27:    C^score = C^score + cd^score
28:      break
29:   end if
30:  end if
31:  end if
32:  end if
33:  end for
34:  end for
35:  C^score = C^score / j   /* j représente le nombre de descripteur de contexte cj dans Csi */
36: return C^score
37: end procedure
```

Figure 7: Context Descriptor Procedure

In our experiments, we propose two matchmaking implementations, corresponding to two distinct service discovery processes:

- ✓ The *I/O Matchmaker* uses references to input and output information provided by the user to proceed with the services selection.
- ✓ The C/FRT Matchmaker implements our service discovery process with support for the context and the requirement. For this, the C/FRT Matchmaker uses the OWL-SRC API, the Jena framework [24] and the Pellet reasoning engine [25]. For each available service in the service repository, the C/FRT Matchmaker calculates a matching score according to the user's requirement and context based on the extended service description. This class requires two other classes, namely *ContextMatching* and *RequirementMatching*. The separation between these two elements enables the evaluation of each component and analysis of their impact on the selection mechanism.

The evaluation of the different service discovery methods was carried out on a semantic repository containing an extended set of service descriptions, taken from the OWL-S service retrieval test collection OWLS-TC4 which contains the descriptions of 1083 Web services from 9 domains. Among the available areas, we have chosen the services in the field of travel. This domain contains around 600 descriptions of services, which have been enriched with contextual information and the elements representing a requirement. As part of our experiments, we deployed our service selection algorithm on a 2.7 GHz Intel Core i7 processor machine with 8 GB memory.

We have described situations where the user requirement elements are not described in the requirement ontologies while there is in the service repository a set of services capable of satisfying this requirement in user's current context (refer to Table 3). For example, in R4, the *Function*, *Resource* and *Transaction behavior* elements of the "*Book-up Lodge*" requirement may not be described in the *Function*, *Resource* and *Transaction behavior* ontologies. However, there is a set of services capable of meeting this requirement, including those that meet the "*Reserve Accommodation _with Compensation op"* requirement which is similar to the R4 requirement (i.e., cancel.hasSynonym = compensate). In R3, the "*fourStarHotel*" resource represents a plug-in of the resource "*Accommodation*". Similarly, for R6, the function "*Locate*" represents a hyponym of the function "*Acquire*". Thus, transactional services that meet the "*Acquire destination_with Cancellation op*" and "*Acquire Surfingdestination_with Cancellation op*", for example may not be selected.

The request processing time was measured by varying the number of services in the services repository between 10 and 600 services, which is indicated in figure 8 by the abscissas axis "Service repository size". The results show that the response time of the selection algorithm follows a polynomial trend. This allows us to assert that the service selection process is implemented at a satisfactory scaling-up level. In more detail, figure 8 presents the comparison between the three service selection algorithms used in this experiment: (i) a service selection mechanism based only on the service's inputs and outputs implemented by the *I/OMatchFacade* class), thus representing a purely functional view; (ii) a contextual service selection mechanism, based only on the validity context of the user implemented by the *FuntionResourceMatchFacade* and *ContextMatchFacade* classes, in which the transaction behavior driven selection is disabled; and (iii) the service selection mechanism driven by the requirement and the context implemented by the *C/FRTMatchFacade* class.

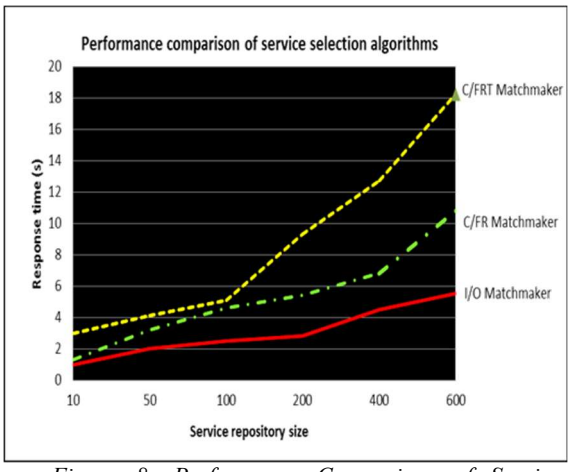

Figure 8: Performance Comparison of Service Matchmaking Mechanisms

7. DISCUSSION

In the experiments that we have carried out, we analyzed the scalability of service selection mechanisms through the average processing time, when we vary the number of services available in the service repository.

In figure 8, we observe that the selection mechanism guided by requirement and context (i.e., C/FRT Matchmaker) has a higher average response time than other algorithms. This difference especially when compared to the I/O algorithm, is not worrying and remains reasonable. We can notice that although we have increased the number of services more than sixty times, the response time increased only six times. However, the performance of the C/FRT service selection mechanism depends on the processing time of the user's transaction behavior requirements and its current context. For example, the evaluation of the user situation described by R5 shown in Table 3, does not take much response time compared to other requests. Indeed, in the case of R5 evaluation, the C/FRT service selection mechanism only processes the Function and Resource elements and does not proceed to the Transaction behavior matching, since the service does not expose transactional properties. Whereas, in the case of evaluating other requests, the C/FRT service selection mechanism deals with requirement and contextual matching, since the requirement can be satisfied by a set of services. For instance, the evaluation of R4, R6, R7 and R8 will take significantly longer to execute than other

requests. This is due to the richness of the requirement elements in the correspondence ontologies on the one hand (i.e., several transaction behaviors which are more specific and/or more generic than the requirement transaction behavior), and on the other hand, to the complexity of the contextual description.

In short, we summarize the characteristics of user requests in Table 4. Performance metric is determined in relation to the average execution time of the service selection algorithm to select the most appropriate service ranging from ++ (i.e. lower execution time) to -- (i.e. higher execution time).

The C/FRT algorithm succeeds in selecting all or almost all of the services that match the user's requirement and context, and this with a low false-positive rate. However, this is only valid in the case of a complete and rich ontologies description and a proper setting of the matching thresholds in the service selection algorithm. Hence, the results analysis demonstrates the reliability of the proposed service discovery process. We believe that the proposed mechanism allows selecting the best service that meets the user's needs, thanks to its hybrid approach of functional and transactional behavior, which is more transparent for the user, and to the use of context that filters the services which are valid (i.e., the validity context of the service vis-à-vis the user).

Table 3: Succinct Illustration of Complex Situations of Requirements in a Given Context.

User Request	Requirement			Current Context
	Function	Resource	Transaction behavior	
R1	Book	Flight	Acquisition op	Disconnected mode Medium bandwidth Location country: Morocco Time: Morning
R2	Reserve	Airline ticket	Acquisition op Compensation op	Connected mode Communication cost : 25% of communication price Location country: Morocco
R3	Reserve	four StarHotel	Acquisition op Replay op	Connected mode Available memory Delay between attempts :1day Number of attempts : 3
R4	Book-up	Lodge	Reservation op Acquisition op Cancellation op	Disconnected mode Full memory Timeout: 12 hours Time: Evening Location city: Marrakech
R5	Find	Guide	--	Communication cost : 15% of communication price Service rate: 8.8
R6	Locate	Surfing destination	Reservation op Acquisition op Cancellation op	Disconnected mode Service rate: 6.5 Timeout: 12 hours Location city: Dakhla
R7	Reserve	Train ticket	Reservation op Acquisition op Cancellation op	Connected mode Low bandwidth Timeout: 12 hours Price: cheap
R8	Book	Ski session	Reservation op Acquisition opCancellation op	Connected mode Low bandwidth Timeout: 12 hours Age: adult users Skill levels: 4 to 6 Location city: Ifrane

Table 4. Performance Metric according to Context and Requirement Concept Complexity

User Request	Missing Concept in the ontology	Requirement Complexity	Context Complexity	Performance Metric
R1		✓ *(Function)*		++
R2		✓ *(Function)*	✓	+
R3		✓ *(Resource)*	✓	++
R4		✓ *(Transaction behavior)*	✓	-
R5	✓	✓ *(Function & Resource)*		--
R6		✓ *(Resource)*	✓	++
R7		✓ *(Transaction behavior)*		+
R8		✓ *(Transaction behavior)*	✓	-

8. CONCLUSION

In this paper, we presented a context-driven transactional services selection by introducing a new selection mechanism "CT2S" based on a semantic matching algorithm. The aim is to classify the services according to their contextual and transactional information, and to select the best service that suits user's request. The analysis of different selection algorithms joint comparison demonstrates the interest of the service selection mechanism proposed in this article. We believe that the proposed mechanism allows to select the service that best corresponds to the user's needs due to its transactional approach, which is more transparent for the user, and to the use of context which limits services to valid ones. However, it is important to note that we can only obtain this good quality results if the system designer establishes from the start a rich description of the services available and the different ontologies used. In addition, this implementation represents a first version used to validate our service selection process, which we plan to optimize in the short term by using services parallel processing.

In our future research, many issues still need to be resolved to fully automate the functional comparison of transactional services. The Requirement and Context attributes type described in our OWL-SRC semantic model aim to expose the two aspects of a service, in particular for discovery purposes. Thanks to the OWL-SRC extension that we offer, a transactional service can be discovered either by the requirement that it can satisfy, or by the context associated with that requirement. In addition to these aspects, a third aspect should be explored which is the service composition described in OWL-SRC.

REFERENCES:

[1] Karlsen R. "An adaptive transactional system - framework and service synchronization", *International Symposium on Distributed Objects and Applications (DOA)*, Catania, Sicily, LNCS, vol. 2888/2003, pp:1208-1225, 2003.

[2] Santos N., Veiga L., and Ferreira P. "Transaction policies for mobile networks", *5th IEEE International Workshop on Policies for Dist. Systems and Networks*, 2004.

[3] Rouvoy R., Serrano-Alvarado P., and Merle P. "Towards Context-Aware Transaction Services", *Proceedings of the 6th International Conference on Distributed Applications and Interoperable Systems (DAIS'06),* Bologna, Italy, Lecture Notes in Computer Science, Springer-Verlag, vol. 4025, pp. 272-288, 2006.

[4] Ettazi W., Hafiddi H., and Nassar M. "CATS-CAE Reflective Middleware Framework for Adapting Context-Aware Transactional Services: Using a Hybrid Policy-Based Approach", *International Journal of Web Services Research (IJWSR)*, Volume 17, Issue 2, 2020.

[5] Yu T., Zhang Y., and Lin K.J. "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints". *ACM Trans. Web* pp.1, 6. 14, 26, 35, 60, 64, 77, 2007.

[6] Schilit B., Adams N., and Want R., "Context-aware computing applications*", Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, California, pp. 85-90, IEEE Computer Society Press, 1994.

[7] Dey A. K., and Abowd G. D., "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", *Human-Computer Interaction (HCI) Journal*, Vol 16, pp 2-4, 2001.

[8] Paolucci M.., Kawamura T., Payne Terry R., and Sycara K. "Semantic Matching of Web Services Capabilities", *Proceedings of the 1st International Semantic Web Conference*. Springer LNCS, volume 2342, Sardinia, Italy, 2002.

[9] Klusch M., Kapahnke P., and Zinnikus I. "Hybrid Adaptive Web Service Selection with SAWSDL-MX and WSDL-Analyzer", *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research*

and Applications, (Berlin, Heidelberg: Springer-Verlag), pp. 550–564, 2009.

[10] Martin D., Burstein M., Mcdermott D., Mcilraith S., Paolucci M., Sycara K., Mcguinness D.L., Sirin E., and Srinivasan N. "Bringing Semantics to Web Services with OWL-S". *Journal World Wide Web* 10(3), pp. 243–277, 2007.

[11] Suraci V., Mignanti S., and Aiuto A. "Context-aware Semantic Service Discovery". *In Mobile and Wireless Communications Summit, 16th IST*, pp. 1–5, 2007.

[12] Ben Mokhtar, S., Preuveneers D., Georgantas N., Issarny V., and Berbers Y. "EASY: Efficient semAntic Service discovery", *pervasive computing environments with QoS and context support. Journal of System and Software*, 81(5), pp. 785–808, 2008.

[13] Toninelli A., Corradi A., and Montanari R. "Semantic-based discovery to support mobile context-aware service access". *Computing Communications*, 31(5), pp. 935–949, 2008.

[14] Ying Y., Zhang B., Zhang X., and Zhao Y. "A Self-healing composite Web service model", *Services Computing Conference, APSCC. IEEE Asia-Pacific*, pp. 307–312, December, 2009.

[15] El Haddad J., Manouvrier M., Ramirez G., and Rukoz M. "QoS-driven selection of web services for transactional composition". *IEEE International Conference on Web Services ICWS'08*, pp. 653-660, 2010.

[16] Zaremski A.M. and Wing J.M. "Signature matching: a tool for using software libraries". *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 4(2), pp. 146–170, 1995.

[17] Vanrompay Y., Kirsch-Pinheiro M., and Berbers Y. "Service Selection with Uncertain Context Information", *Service-Oriented Systems and Non-Functional Properties: Future Directions, S. ReiffMarganiec, and M. Tilly, eds. (IGI Global)*, pp. 192–215, 2011.

[18] Petit M., Ray C., and Claramunt, C. "A contextual approach for the development of GIS: Application to maritime navigation", *6th International Symposium of Web and Wireless Geographical Information Systems (W2GIS). J. Carswell and T. Tezuka (eds.). Springer-Verlag LNCS 4295*, Hong Kong, December 4-5, pp 158-169, 2006.

[19] Fissaa T., Guermah H., El Hamlaoui M., Hafiddi H., and Nassar M. "A Synergy of Semantic and Context Awareness for Service Composition in Ubiquitous Environment". *Computer and Information Science* 11(2): pp.88-98, 2018.

[20] W3C, "OWL-S: Semantic Markup for Web Services", *W3C Member Submission* 22 November 2004.

[21] Kirsch-Pinheiro M., Vanrompay Y., and Berbers Y. "Context aware service selection using graph matching", *2nd Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop (NFPSLA-SOC'08), ECOWS 2008. CEUR Workshop proceedings*, Vol. 411, 2008.

[22] Bhiri S., Perrin O., and Godart C. "Ensuring required failure atomicity of composite web services", *Proceedings of the 14th international conference on World Wide Web, WWW'05*, New York, NY, USA, pp. 138–147, 2005.

[23] Schuldt H., Alonso G., and Schek H. "Concurrency control and recovery in transactional process management", *Proceedings of the Conference on Principles of Database Systems*, pp. 316-326, Philadelphia, Pennsylvania, 1999.

[24] Carroll J.J., Dickinson I., Dollin C., Reynolds D., Seaborne A., and Wilkinson K. "Jena: implementing the semantic Web recommendations", *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, (New York, USA: ACM), pp. 74–83, 2004

[25] Parsia B., and Sirin E. "Pellet: An owl dl reasoner", *Proceedings of the International Workshop on Description Logics*, 2004.