Electronic Communications of the EASST
Volume 19 (2009)

Proceedings of the Second International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 2009)

Context-Aware Service Selection with Uncertain Context Information

Yves Vanrompay,Manuele Kirsch-Pinheiro,Yolande Berbers

11 pages

Guest Editors: Romain Rouvoy, Michael Wagner
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Context-Aware Service Selection with Uncertain Context Information

Yves Vanrompay[1]**,Manuele Kirsch-Pinheiro**[2]**,Yolande Berbers**[1]

[1] firstname.lastname@cs.kuleuven.be
Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Heverlee

[2] Manuele.Kirsch-Pinheiro@univ-paris1.fr
Centre de Recherche en Informatique, Université Paris 1 Panthéon-Sorbonne
90 rue de Tolbiac, 75013 Paris, France

Abstract: The current evolution of Service-Oriented Computing in ubiquitous systems is leading to the development of context-aware services. These are services whose description is enriched with context information related to the service execution environment and adaptation capabilities. This information is often used for discovery and adaptation purposes. However, in real-life systems context information is naturally dynamic, uncertain and incomplete, which represents an important issue when comparing service description and user requirements. Uncertainty of context information may lead to an inexact match between provided and required service capabilities, and consequently to the non-selection of services. In order to handle uncertain and incomplete context information, we propose a mechanism inspired by graph-comparison for matching contextual service descriptions using similarity measures that allow inexact matching. Service description and requirements are compared using two kinds of similarity measures: local measures, which compare individually required and provided properties, and global measures, which take into account the context description as a whole. We show how the proposed mechanism is integrated in MUSIC, an existing adaptation middleware, and how it enables more optimal adaptation decision making.

Keywords: service selection, SOA, context-aware computing

1 Introduction

The term Ubiquitous Computing, introduced by Weiser [1], refers to the seamless integration of devices into users' everyday life [2]. This term represents an emerging trend towards environments composed of numerous computing devices that are typically mobile or embedded and that are connected to a network infrastructure composed of a wired core and wireless edges [3]. In pervasive scenarios foreseen by Ubiquitous Computing, context awareness plays a central role. Context can be defined as any information that can be used to characterize the situation of an entity (a person, place, or object considered as relevant to the interaction between a user and an application) [4]. Context-aware systems are able to adapt themselves to their environment,

aiming at optimizing QoS for the user and resource consumption by taking context information into account.

The dynamicity of pervasive environments encourages the adoption of a Service Oriented Architecture (SOA). Service-Oriented Computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications [5]. The key feature of SOA is that services are independent entities, with well-defined interfaces, that can be invoked in a standard way, without requiring the client to have knowledge about how the service actually performs its tasks [6]. Such loose coupling fits the requirements of highly dynamic pervasive environments, in which entities are often mobile, entering and leaving the environment at any moment. The adoption of SOA in pervasive environments is leading to the development of "context-aware" services. Context-awareness becomes a key feature necessary to provide adaptable services, for instance when selecting the best-suited service according to the relevant context information or when adapting the service during its execution according to context changes.

However, in ubiquitous environments, context information is naturally uncertain and incomplete. Uncertain and incomplete context information may prevent perfect matches between required and provided properties, which may lead to the non-selection of a service. In other words, when executing in pervasive environments, service matching mechanisms have to deal with the question: how to reduce problems related to mismatching between contextual conditions related to the execution of a service and the current context? Service selection mechanisms have to cope with this issue: if some needed context information is missing or if it is uncertain, service selection still has to proceed and choose a corresponding service that best matches the current situation. These mechanisms should take into account uncertainty when ranking context-aware services. Even if context requirements seem to match, if context information is too uncertain, this match should be considered as proportionally imprecise. Consequently, a service that is well ranked can receive a lower rank since this ranking is based on uncertain information. Then if this ranking is calculated by a utility function, a service with medium utility can be chosen over one with high utility, if the latter utility has a high uncertainty value associated with it.

We propose in this paper a mechanism inspired by graph-matching to overcome this issue. The matching combines local and global similarity measures that compare context elements individually and context information and requirements as a whole. Such measures use uncertainty associated as metadata with context information in order to better rank discovered services. This paper is organized as follows. Section 2 gives an overview of the MUSIC middleware. Then, section 3 presents the approach and section 4 shows how it is integrated in MUSIC using an example GPS service. Section 5 presents related work and we conclude in section 6.

2 MUSIC middleware

The service selection approach proposed in this paper is part of a larger initiative, the MUSIC Project. The MUSIC Project [14] is a focused initiative aiming at the development of context-aware self-adapting applications. The main target is to support both the development and runtime management of software systems that are capable of being adapted to highly dynamic user and execution context, and to maintain a high level of usefulness across context changes. MUSIC adopts a service-oriented approach in which modeling languages allow the specification of

context dependencies and adaptation capabilities. Such adaptation capabilities are based on the specification, at design time, of multiple variations (implementations) for each component (or service). The selection of the most appropriate variant is performed at runtime by the MUSIC middleware based on the context dependencies associated with each variant and based on the current execution context. *Property predictor functions* express the expected QoS provided by components in a given context. These values are used in a *utility function* to calculate the utility of a variant. Finally, the application is adapted to the variant with the highest utility given the current context.

The MUSIC context modeling approach [16] identifies three basic layers of abstraction that correspond to the three main phases of context management: the conceptual layer, the exchange layer and the functional layer. The *conceptual layer* enables the representation of context information in terms of *context elements*. These provide context information about *context entities* (the concrete subjects the context data refers to: a user, a device, etc.) belonging to specific context scopes. Such *context scopes* are intended as semantic concepts belonging to a specific ontology described in OWL. Moreover, the ontology is used to describe relationships between entities, e.g. a user has a brother. The *exchange layer* focuses on the interoperability between devices. Context data in this layer is represented in XML and is used for communication between nodes. The *functional layer* refers to the implementation of the context model internally in the different nodes.

Additionally, the MUSIC middleware collects context information from a set of context plug-ins [17]. Each plug-in handles a given pair of context entity and scope, periodically updating the corresponding context element. In addition to context values, context plug-ins associate different *metadata* with context elements they observe. These metadata include timeout and other indicators concerning the quality of collected data.

The next section introduces a service selection mechanism, which is integrated in the MUSIC platform and considers uncertainty of context information. As such, this mechanism adheres to the MUSIC principles described above, namely: (i) evaluation of service variants using utility functions; (ii) modeling of context information by means of context entities and scopes; (iii) association of metadata with such context elements. These principles used in the MUSIC middleware form the basis for the proposed service selection approach.

3 Service selection under uncertainty

3.1 Services and MUSIC

The MUSIC project aims at exploiting SOA by allowing MUSIC applications to consider the variability and non-functional properties of context-aware services. In other words, several service implementations can supply the same functional capabilities (with a similar syntax), but with different non-functional context-related properties.

The service selection approach proposed here complements the service selection mechanism used by the MUSIC middleware for selecting the most suitable service among discovered and compatible services. The MUSIC middleware compares context-aware service descriptions and current execution context in order to rank suitable services, taking into account the current situation. Our service selection mechanism assumes that suitable services exist. It is part of a

two-step process in which the first step discovers all services whose functional properties match the functional requirements that are needed. This means our approach (the second step), dealing with non-functional requirements, is employed only after suitable services are discovered. If there are several discovered services able to satisfy a request formulated by a client application, one has to select the service that suits best the current execution context. Such service selection should take into account the fact that context information is naturally uncertain and incomplete.

We focus on non-functional context-related aspects of a service description, assuming the selection process for meeting functional requirements (inputs and outputs) already took place. Functional aspects of a service have the priority and are fulfilled in the first step of the selection process, since mismatching on service input or output may affect correctness and execution flow on both the service and the calling application. Thus, in this paper we focus only on non-functional conditions related to the execution environment of context-aware services.

Service context requirements are modeled following the MUSIC context modeling approach, presented in the previous section. The description of the non-functional service requirements, illustrated in Figure 1, belongs to the exchange level, since it is used for information exchange among different nodes. Thus, context information is described in XML by context elements, which refer to a given entity and scope. Figure 1 illustrates the description of a context-aware service that indicates the conditions to which this service implementation is best suited to (i.e. the contextual situation in which it is most appropriate to call this service). More details about this description can be found in [15].

The matching algorithm compares these descriptions of services with the description of the current context. This matching is inspired on local and global similarity measures used in graph-matching. From a conceptual point of view, the descriptions of the service context requirements and of the current context can be represented as graphs, in which context elements represent nodes and relations between these represent edges. Local measures can be used to compare node to node of these graphs, while a global measure structurally looks at the similarity of the graphs. This general matching algorithm is described in previous work . In this paper, we integrate the theory worked out in [15] in the MUSIC adaptation mechanism that uses utility functions. While in this paper we do not work explicitly with graphs, the analogy with graphs still remains valid.

3.2 Matching with uncertainty

The goal of the matching algorithm is to rank the available services based on their contextual non-functional properties. It compares the context description related to available services with the current execution context. This matching starts with comparing individual context elements from both descriptions (from the context description of the service and from the current context) individually, using local similarity measures. Then the global similarity measure is taken into account by building the *utility function* from the local measures. The results of such global measures, which are utility function values, are used to rank the services for their suitability in the current context.

First we compare the current context values with the service requirements using *local similarity measures*. For the node-to-node (i.e. context element to context element) comparison we use measures as those proposed by *SimPack* [18], depending on the scope of the corresponding context element. For example, for numerical values the measure represents how close the num-

```
− <ctx:context xsi:schemaLocation="http://www.ist-music.org/ContextSchema ContextSchema.xsd ">
  − <ctx:condition>
    − <ctx:contextElement>
        <ctx:hasEntity resource="http://www.ist-music.org/Ontology/ContextModel.owl#concept.entityType.user"/>
        <ctx:hasScope resource="http://www.ist-music.org/Ontology/ContextModel.owl#concept.contextScope.location"/>
        <ctx:hasRepresentation resource="http://www.ist-music.org/Ontology
        /ContextModel.owl#concept.representation.locationDefaultRepresentation"/>
      − <ctx:contextValueSet>
        − <ctx:contextValue>
            <ctx:hasScope resource="http://www.ist-music.org/Ontology
            /ContextModel.owl#concept.contextScope.location.city"/>
            <ctx:hasRepresentation resource="http://www.ist-music.org/Ontology
            /ContextModel.owl#concept.representation.locationDefaultRepresentation"/>
            <ctx:value>Paris</ctx:value>
          </ctx:contextValue>
        </ctx:contextValueSet>
      </ctx:contextElement>
    − <ctx:contextElement>
        <ctx:hasEntity resource="http://www.ist-music.org/Ontology/ContextModel.owl#concept.entityType.user"/>
        <ctx:hasScope resource="http://www.ist-music.org/Ontology
        /ContextModel.owl#concept.contextScope.userprofile"/>
        <ctx:hasRepresentation resource="http://www.ist-music.org/Ontology
        /ContextModel.owl#concept.representation.profileDefaultRepresentation"/>
      − <ctx:contextValueSet>
        − <ctx:contextValue>
            <ctx:hasScope resource="http://www.ist-music.org/Ontology
            /ContextModel.owl#concept.contextScope.profile.category"/>
            <ctx:hasRepresentation resource="http://www.ist-music.org/Ontology
            /ContextModel.owl#concept.representation.profileDefaultRepresentation"/>
            <ctx:value>Tourist</ctx:value>
          </ctx:contextValue>
        </ctx:contextValueSet>
      </ctx:contextElement>
  </ctx:condition>
  + <ctx:state></ctx:state>
</ctx:context>
```

Figure 1: Context description of a service.

bers are lying together, relative to their range. The result of one node-to-node comparison is a *mean* (between 0 and 1) with an uncertainty degree. The *uncertainty* of a context value is given as a *metadatum* of that element between 0 and 1. A context plug-in, when collecting/calculating a given context element, can estimate an uncertainty degree associated with context values. It is the only one that can perform such an estimation, since the context middleware is not directly aware of how data is collected. Thus, when looking at the similarity, we also take into account uncertainty by combining the uncertainty of a current context value with the uncertainty resulting from the prediction of the QoS properties of a service (note that a MUSIC service will run locally on the node). *Incompleteness* of context information is dealt with by taking the average value as mean and the *uncertainty* degree as maximal, i.e. 1.

Thus, the local similarity of the service context requirements with the current context is expressed by the *mean.* A low mean expresses a low similarity and vice versa. In case the uncertainty degree is 0 and the current context value and service context requirement are equal (exact

match), the similarity measure is 1. We call the combination of mean and resulting uncertainty degree the *partial utility* for the context element.

When the partial utilities of each service variant have been calculated, we have to combine them. In other words, based on these partial utilities, we calculate a *global similarity measure* and use a ranking scheme to select the service with the highest overall utility. The *overall utility* is a normalized weighted sum of the different partial utilities. Highest utility is not necessarily the utility with the highest mean because of the uncertainty of the context information: a given utility with a mean that is only a little bit higher than another, but that has much greater uncertainty degree is worse as a result. We prefer to minimize the "risk", which is defined as the probability that an event will occur times the consequences (impact) if it does occur. A higher uncertainty degree expresses more risk that the actual mean value will be different. For example, when looking at the uncertainty degrees expressing the current available memory, we will prefer a lower amount of memory if we are more certain that the amount of memory will actually be available than risking a memory shortage. If the means are equal then of course less uncertainty degree is better. If the uncertainty degrees are equal then higher mean is preferable. In the borderline case, we use *thresholds* for both mean and uncertainty values. For example, let us consider thresholds of 0.1 for mean values and 0.2 for uncertainty values: if two utilities differ less than 0.1 in their mean values, then the utility with the highest mean is the best only if its uncertainty degree is maximum 0.2 higher than the uncertainty degree of the utility with lower mean. If mean difference between the overall utilities is more than 0.1 than the highest is in each case the best utility, whatever the uncertainty degree is. The result of this utility function is then used to rank available variants of a service.

The following section presents an example of a route planner service ("GPS service") with several variants. We explain how the partial utility functions and the overall utility function are calculated. We also give examples of how uncertainty can be given as metadatum of a context element or even can be the consequence of a property predictor function (which calculates the expected QoS of a component in a given context).

4 Case study

In this section, we illustrate our approach through a case study. For this, let us consider a route planner service ("GPS service") that calculates the route for a user based on GPS information. This service has several implementations, which vary according to two main criteria (*non-functional properties*): *route quality* and *response time*. Such non-functional properties correspond to user preferences represented, according to MUSIC context modeling approach, as part of the user context, and more precisely, as part of the user profile (i.e. context elements with scopes routequality and response, as indicated in Figure 2).

Additionally, we consider the following observed context scopes: available memory resources, GPS signal strength (0-100), and light conditions (0 or 1). For each context scope observed, a partial utility function is proposed in order to compare locally context elements corresponding to this scope. Examples of such partial utility functions are proposed in Figure 3.

In addition to these partial utility functions, property predictor functions can be defined for each service variant. For example, the route planner service proposes variants based on the

Criteria for GPS service variants:

- route quality requested by application varies between 0 and 1
- time requested by application varies between 4 and 20
- context.routequality and context.response are user preferences

Figure 2: Non-functional properties determining service variants.

Partial utility functions: $\langle mean, uncertainty \rangle$

memory-utility = if (context.availablememory > service.neededmemory) 1
else 0

routequality-utility = if (context.routequality) > service.routequality) 1
else 1 - (service.routequality - context.routequality)

response-utility = if (context.response > service.response) 1
else 1 - (service.response - context.response)

light-utility = if (context.light and service.light) or
(!context.light and !service.light) 1
else 0

Figure 3: Partial utility functions evaluating memory, route quality, response time and light.

response time information, which are calculated using the GPS signal strength, as well as the available memory observed by the corresponding context plug-ins (Figure 4).

In order to rank available services, the service selection mechanism considers mean values obtained with previous functions in a overall utility function. Thus, considering the route planner service, let us suppose that the following context elements are currently available: *context.availablememory*, *context.routequality*, *context.response* and *context.light*. *Service.neededmemory*, *service.routequality*, *service.response* and *service.light* are non-functional requirements of this service, obtained from the XML context description file associate with service variants. It is worth noting that we indicate observed context values by *context.scope* and service values by *service.scope*.

Then, supported variation points are light (2 variants), memory needed (3 variants with different response times, as discussed in Figure 4) and offered route quality (3 variants), which means that, in total, there are 18 service variants. We need to select the service variant with the highest utility. However, *service.response* element is uncertain because *context.signalstrength* measured value in property predictor function is uncertain (e.g. since signal strength fluctuates and measurement is done by imprecise software). This means that *context.signalstrength* is a context value with a *mean* value and an *uncertainty* as metadatum.

The *response-utility* function uses the mean value of the previous step to compute the *response-utility* and transforms the uncertainty value into an uncertainty degree. As a result, the output of the *response-utility* is a *mean* and an *uncertainty degree*, which is input for the overall utility function. The similarity of current context value and service requirement is used as an extra

Property predictor functions for service.response:

- low-response-service:
  service.response = if (context.signalstrength >80) 5
  else 5 + ((80 - context.signalstrength) / 80) * 20
  This variant needs minimal 10 units memory

- medium-response-service:
  service.response = if (context.signalstrength >50) 5
  else 5 + ((50 - context.signalstrength) / 50) * 20
  This variant needs minimal 40 units of memory

- high-response-service:
  service.response = if (context.signalstrength >20) 5
  else 5 + ((20 - context.signalstrength) / 20) * 20
  This variant needs minimal 100 units of memory

Figure 4: Examples of property predictor functions.

weight in the utility function. The *overall utility function* for the case study considered here is presented in Figure 5.

The result of each *partial utility function* is a *mean* with an *uncertainty degree*. The means are filled in as values of the *overall utility function*. This gives a value that is the "*utility mean*". The utility also has an uncertainty degree: this is a function of the *uncertainty* degree of each partial utility function weighted according to the weights in the overall utility function. Service variants are ranked based on these values (utility mean and uncertainty), as explained in Section 4.2: for "normal" cases, variants with higher mean will be better ranked than variants with a lower mean value. However, in the borderline cases, when the difference between values associated with two variants is lower than a predefined *threshold* (0.2 in this case), the service variant with a lower uncertainty degree will be better ranked than a variant with a higher one.

5 Related work

A growing interest in context-aware services can be observed in the literature. For instance, several European projects are focusing on Service-Oriented Computing [7], and context-awareness appears as a crosscutting issue for these works. According to Tonielli et al. [8], in pervasive scenarios, users require context-aware services that are tailored to their needs, current position, execution environments, etc. According to Suraci et al. [9] user and service entities have requirements on context information they need in order to work properly. A user may have requirements on the context of the service he is looking for (availability, location...) and on the context pro-

Overall utility function:

utility(mean) = if (memory-utility(mean) = 0) 0
else (0,25 * light-utility(mean) + 0,50 * response-utility(mean)
+ 0,25 * routequality - utility(mean))

utility (uncertainty-degree) = if (memory-utility(mean) = 0)
memory-utility(uncertainty-degree)
else (0,25 * light-utility(uncertainty-degree)
+ 0,50 * response-utility(uncertainty-degree)
+ 0,25 * routequality - utility(uncertainty-degree))

Figure 5: Example of overall utility function.

vided by the environment (wireless connection...). A service can require the user to provide specific context information (location, terminal capabilities...) and the environment to provide context information too (network QoS...).

Most research tracks on context-aware services focus on how to describe and discover these services [7]. For instance, Suraci et al. [9] propose a semantic modeling of services in which service profile descriptions in OWL-S are enriched with a "context" element pointing to this required context information. Ben Mokhatar et al. [10] propose the use of ontologies in OWL-S for the semantic description of functional and non-functional features of services in order to automatically and unambiguously discover such services. Other works, such as [11], focus on service matching algorithms considering context information. For instance, Klusch et al. [11] propose a service matching algorithm which combines reasoning based on subsumption and similarity measures for comparing inputs and outputs of service description and user request. Reiff-Marganic et al. [12] propose a method for automatic selection of services based on non-functional properties and context.

However, works cited above do not consider inexact matching caused by incomplete or uncertain context information. Context information is naturally dynamic and uncertain: it may contain errors, be out-of-date or even incomplete. Uncertainty in context information is traditionally handled by appropriate models, such as proposed by Chalmers et al. [13], who represent context values by intervals or sets of symbolic values. Other approaches such as [19] and [20] deal with uncertainty using fuzzy logic. However, they do not concentrate on inexact service matching. Service selection algorithms ought to consider uncertainty represented in their models. These algorithms should take into account uncertainty when ranking and selecting services. Nevertheless, algorithms proposed in previously cited works do not consider uncertainty of context information, assuming that such information is precise enough. We argue that uncertainty on context information must be considered when ranking and selecting available services.

6 Conclusions

In this paper, we proposed a service selection mechanism that takes into account uncertainty of context information when ranking service variants. This mechanism integrates theory worked out in the MUSIC project that uses utility functions to evaluate available variants. Inspired by this theory, the service selection mechanism uses partial utility functions combined to overall utility functions in order to compare the context-related requirements of each service variant with the current context values. Such functions are based on the analysis of mean values and uncertainty degrees, allowing the MUSIC middleware to rank available variants considering not only context values, but also their mean and estimated uncertainty. As future directions, we expect to test the performance of the mechanism with traditional service selection mechanisms initially proposed in MUSIC (that do not deal with uncertain context information) in ubiquitous environments. This will allow us to experimentally evaluate the threshold values that need to be set concerning the uncertainty degree and utility.

Acknowledgements: The authors would like to thank their partners in the MUSIC-IST project and acknowledge the partial financial support given to this research by the European Union (6th Framework Programme, contract number 35166).

Bibliography

[1] Weiser M., The computer for the 21st century, Scientific American, vol. 66, 1991.

[2] Baldauf M., Dustdar S., Rosenberg F., A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing, vol. 2, n.4, 2007, pp. 263277

[3] Moran T., Dourish P., Introduction to this special issue on context-aware computing. Human-Computer Interaction, vol. 16, n. 2-3, 2001, pp. 8795

[4] Dey A., Understanding and using context. Personal and Ubiquitous Computing, vol. 5, n. 1, 2001, pp. 4-7

[5] Papazoglou M. P., Georgakopoulos, D., Service-Oriented Computing, Communication of ACM, vol. 46, n. 10, Oct. 2003, pp. 24-28

[6] Issarny V., Caporuscio M., Georgantas, N., A Perspective on the Future of Middleware-based Software Engineering. Briand, L. and Wolf, A. (Eds.), Future of Software Engineering 2007 (FOSE), ICSE (International Conference on Software Engineering), IEEE-CS Press.

[7] Patouni, E.; Alonistioti, N., Polychronopoulos, C. (eds.), Service Adaptation over Heterogeneous Infrastructures, White Paper, 2008. http://www.opuce.tid.es/Publications.htm

[8] Toninelli, A.; Corradi, A., Montanari, R., Semantic-based discovery to support mobile context-aware service access, Computer Communications, vol. 31, n 5, March 2008, pp. 935-949.

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

[10] Ben Mokhtar, S.; Kaul, A.; Georgantas, N. and Issarny, V., Efficient Semantic Service Discovery in Pervasive Computing Environments, Proceedings of the ACM/IFIP/USENIX 7th International Middleware Conference (Middleware'06), 2006

[11] Klusch, M.; Fries, B. and Sycara, K., Automated semantic web service discovery with OWLS-MX, Proceedings of the 5th International joint conference on Autonomous agents and multiagent systems (AAMAS '06), ACM, 2006, 915-922

[12] Reiff-Marganiec, S., Qing Yu, H., Tilly, M., Service Selection based on Non-Functional Properties, NFPSLA-SOC07 Workshop at The 5th International Conference on Service Oriented Computing (ICSOC2007), Sept. 17 2007, Vienna, Austria.

[13] Chalmers, D.; Dulay, N. and Sloman, M., Towards Reasoning About Context in the Presence of Uncertainty, 1st international workshop on advanced context modelling, reasoning and management, Nottingham, UK, September 2004

[14] MUSIC Consortium, Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environments (MUSIC), Website: http://www.ist-music.eu/

[15] Kirsch-Pinheiro M., Vanrompay Y., 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), at ECOWS 2008, Dublin, Ireland, November 12, 2008. CEUR Workshop proceedings, vol. 411.

[16] Reichle, R., Wagner, M., Khan, M.U., Geihs, K., Lorenzo, L., Valla, M., Fra, C., Paspallis, N., Papadopoulos G.A. A Comprehensive Context Modeling Framework for Pervasive Computing Systems, 8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS), 4-6 June, 2008, Oslo, Norway, Springer.

[17] Paspallis, N., Rouvoy, R., Barone, P., Papadopoulos, G., Eliassen, F., Mamelli, A., A Pluggable and Reconfigurable Architecture for a Context-aware Enabling Middleware System, M. Little, A. Montresor, G. Pavlik (Eds.), 10th International Symposium on Distributed Objects, Middleware, and Applications (DOA'08), LNCS 52331, Monterrey, Mexico, 2008, Springer-Verlag, pp. 553570.

[18] http://www.ifi.uzh.ch/ddis/simpack.html

[19] Chauvel, F.; Barais, O.; Borne, I.; Jezequel, J.-M. Composition of Qualitative Adaptation Policies, ASE 2008. 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008, pp. 455-458

[20] Jiang,, Ying and Dong,, Hui, Uncertain Context Modeling of Dimensional Ontology Using Fuzzy Subset Theory, SUM '08: Proceedings of the 2nd international conference on Scalable Uncertainty Management, 2008, pp. 256-269