Service Interoperability on Context Level in Ubiquitous Computing Environments

Thomas Strang[1], Claudia Linnhoff-Popien[2]

[1]German Aerospace Center (DLR)
Institute for Communikations and Navigation
Muenchener Str. 20
D-82230 Wessling/Oberpfaffenhofen, Germany
Email: thomas.strang@dlr.de

[2]Ludwig-Maximilians-University Munich (LMU)
Institute for Computer Science
Oettingenstr. 67
D-80538 Munich, Germany
Email: linnhoff@informatik.uni-muenchen.de

ABSTRACT

Within distributed systems, interoperability is a key feature enabling the interaction between distributed components. This paper elaborates several approaches of achieving interoperability on different levels with a focus on service interoperability. The analysis of existing approaches motivates the introduction of the context level as a new level of service interoperability. It is shown how context relates to interoperability and why it is useful to deal with contextual service interoperability on a separate level. In particular, ubiquitous computing systems benefit from a procedure called dynamic context association, which supports service provider and service user in gaining access to diverse and rapidly changing context information. A new XML based Context Ontology Language (CoOL) which specifies a common understanding of the relations between services and context models is outlined.

Keywords: Interoperability, Context Awareness, Ubiquitous Computing, Context Ontology Language

I. INTRODUCTION

Modern IT infrastructures are designed following the principle of distributed systems currently, where a set of individual computers are connected via network. In this way they are available as a coherent system for users and different applications [1]. One of the most important properties of distributed systems is the cooperation of different components to achieve a common aim. It is especially remarkable, that the differences of the participating components with respect to hardware, network or software, as well as the kind of communication between the components, are adapted by the system and hence, hidden from the user. By doing so, users and applications are able to interact in a consistent and uniform way within a distributed system.

Any cooperation requires a common understanding of the cooperating components about the data exchanged and the services provided. Thereby the so called *interoperability* between two or more components is guaranteed by applying a common specification. Wegner [2] defines interoperability as the "ability of two or more software components to cooperate despite differences in language, interface, and execution platform". But as we will show in section II, this is a very general definition of interoperability. Within the past 20 years the issue of interoperability has been handled on several levels, but the proposed solutions are not able to fulfill all of the requirements arised. Particularly the context of an application or service is covered imperfectly in existing interoperability considerations, which is our motivation of introducing a new context level in section III. In this section we introduce the terminology used by us to describe the context, how information regarding the context may be sensed and processed and why particularly Ubiquitous Computing environments benefit from taking context into account. Contextual service interoperability is reviewed from two different perspectives, compatibility and substitutability, in section IV. It is motivated, why it is necessary to have a context ontology language to specify a common understanding of the relations between services and context models, before we conclude our work in section V with a summary and an outlook.

II. INTEROPERABILITY IN DISTRIBUTED SYSTEMS

Interoperability is an objective addressed very early during the design, development and enhancement of distributed systems, and requires usually some kind of middleware or platform during runtime. Interoperability disburdens the development with respect to the reusability of components, which are usually developed at different places and times by different persons.

Already at the beginning of the evolution away from centralized approaches towards distributed systems consisting of a network of connected powerful personal computers in the middle of the 80's, the first step of the ladder of interoperability evolution had been climbed with the introduction of the concept of the *Remote Procedure Call (RPC)*. An RPC allows to make a call to a procedure which is located on another machine than the caller instance by keeping this mechanism itself transparent. Any input parameter of a function is serialized (marshalling), any output or return parameter of a function call is deserialized (unmarshalling) by the platform providing the RPC functionality. The necessary routines and formats (e.g. parameter encoding rules) have been specified and published, enabling the interoperability of hardware components of different hardware vendors (*plattform interoperability*).

At the end of the 80's the next step on the ladder of interoperability evolution was climbed with CORBA's *Interface Definition Language (IDL)*. By the combination of distributed technologies as an instrument of the abstraction from the communication layers and the object orientation as the concept of encapsulation and reusability it could be achieved, that applications, which are developed in different programming languages, running on unequal hosts and operating systems using different

network protocols, are interoparabel to each other (*programming language interoperability*).

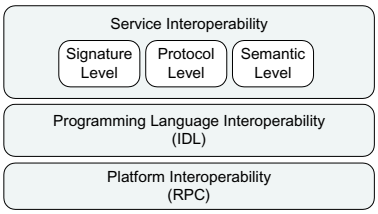

Fig. 1
CLASSIC LEVELS OF INTEROPERABILITY

Research on the issue of interoperability has been further intensified in the 90's and has been honed particularly on the level of applications and services. In literature, *service interoperability* is usually subclassified to *signature level*, *protocol level* and *semantic level* [3], [4], [5] (see also Fig. 1):

- The syntax of the interface of any service is described on the signature level. This encloses usually the *name* of any operation as well as *type* and *sequence* of any parameter of the interfaces. Popular languages for interface specification are CORBA's *Interface Definition Language (IDL)* or the Web Service's *Web Service Definition Language (WSDL)* [6]. The standardisation of the mechanisms to ensure interoperability on the signature level benefit from the greatest progress at the time being.
- Interoperability on the protocol level is aspired by definition of the *relative order*, in which the methods of a service are called, respectively in which a service calls the methods of another service on his part, as well as any *blocking conditions*. Whereas the first ideas on interoperability on the protocol level have been published already 1997 by Yellin and Strom [7], appropriate approaches currently gaining a renaissance by the term *Web Service Choreography Interface (WSCI)* [8]. Also on the protocol level can be seen access rules, defining the conditions of authorized service access [9]. A comprehensive analysis of the protocol level can be found for instance in [10].
- The problem of a divergent understanding and interpretation is attempted on the semantic level, because the information regarding the semantic of a component is usually not covered by any interface or protocol description. Heiler presented in [3] the sample, that according to a study "the probability that two database designers, even domain experts, will chose the same element names for the same data is only between seven and eighteen percent". Often the developer and the user of a component have divergent views of the possible fields of application and the comprehension of the component's services. Thus, an ontology [11] is required, which tries to specify a common understanding by only using a defined terminology for describing the *semantic meaning*. There are still many open questions in the semantic level. XML based languages like DAML+OIL [12] or its successor DAML+S [13], [14], both designed in the framework of the *Semantic Web*, may be capable to fulfill some of the requirements of that level. A comprehensive list of publications regarding the semantic level can be found at [15].

The next sections will discuss, to what extend specifications on a fourth level, the *context level*, are advantageous and valuable to describe the interoperability of components (with a focus on services).

III. CONTEXT LEVEL

After having done a deep inspection of a wide range of past and current research publications and proposals we arrived at the conclusion, that when considering service interoperability at *signature*, *protocol* and *semantic level*, the *context* is covered imperfectly. Already S. Heiler stated in [3] that "interesting semantic information is context-dependent". But information regarding the context are considered on the semantic level inadequately [10], because even if on the semantic level the (internal) *behaviour* of a service can be described for instance by the use of *pre/post conditions* [16], [13] or *abstract type frameworks* [17], the relation to the (external) situation is usually not covered in literature dealing with interoperability. Thus we propose to bundle reflections with respect to the context in its own layer, the so called *context level*, (see also Fig. 2).

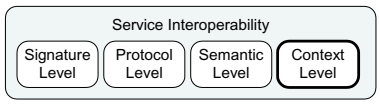

Fig. 2
EXTENDED LEVEL OF SERVICE INTEROPERABILITY

Because of the fact that the terms context and situation in current publications is used in various ways it is necessary to define the terminology used by us:

- A *context information* is any information which can be used to characterize the state of an entity concerning a specific aspect.
- An *entity* is a person, a place or in general an object.
- An *aspect* is a classification, symbol- or value-range, whose subsets are a superset of all reachable states.
- A *context* is the set of all context information characterizing the entities relevant for a specific task in their relevant aspects.
- An *entity is relevant* for a specific task, if its state is characterized at least conceirning one relevant aspect.
- An *aspect is relevant*, if the state with respect to this aspect is accessed during a specific task or the state has any kind of influence on the task.
- A system is *context aware*, if it uses any kind of context information before or during service provisioning.
- A *situation* is the set of all known context information.

Our definition of the terminology is in the first items close to the one given by Dey [18]. Compared to Dey, we try to be more concrete when using the term *relevance* by distinguish between an entity being relevant or an aspect being relevant. The definitions above differ from most other definitions of context (e.g. [18], [19], [20]) by introducing the terminology of an *aspect*. Think of an *aspect* primarily as an axis of discrete or continous values, where a concrete *context information* is an instance of that aspect, where the state of an entity is characterized "softly"

by one or more elements of the allowed value range. For instance if the aspect is based on continous values like an interval of real numbers, a valid context information with respect to that aspect may be a probability density function (PDF) [21]. For each single *entity*, the relation between our definition of *context* and multiple *aspects* fits Couderc's image of a "kind of cursor in a multi-dimensional information space" [22], [23]. Figure 3 shows an example of the usage of the terminology introduced above, where a specific context information (geographical position) with respect to a specific aspect (Gauss-Krüger coordinates) characterizing a specific entity (mobile phone) is expressed in an XML instance document.

```
<instance
  xmlns="http://demo.heywow.com/schema/cool"
  xmlns:a="http://demo.heywow.com/schema/aspects"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <contextInformation>
    <entity system="urn:phonenumber">
      +49-179-1234567
    </entity>
    <characterizedBy>
      <aspect name="GaussKruegerCoordinate">
        <observedState xsi:type="a:o2GaussKruegerType">
          367032533074
        </observedState>
        <units>10m</units>
      </aspect>
      <certaintyOfObserver>90<certaintyOfObserver>
    </characterizedBy>
  </contextInformation>
</instance>
```

Fig. 3
EXAMPLE OF A CONTEXT INFORMATION INSTANCE DOCUMENT

According to the definition above, the situation of a service is the set of all known context information being available before or during a service's execution. Under these circumstances it is irrelevant, whether the context information is provided by the service itself, the service provider or any other element of the infrastructure. Here we have an important distinctive feature to several approaches of the semantic level, where merely access is given to the *internal state* of an object, especially if using pre/post conditions [10]. We are currently designing and implementing a system which allows one to associate context information at runtime dynamically to any object of interest. In our system *context observers* are responsible for acquiring sensor data, as well as processing and cultivating that data to represent context information. Context information are associated to real objects by *shadow objects* (see also Fig. 4).

A shadow object is a representative of an entity which may, but does not require to have a network accessible object instance. An example: If the object is a thermometer, the associated shadow object contains context information regarding some temperature aspect, although the thermometer itself may not have any electronic interface. It is up to the assigned context observer how to obtain the context information, for instance by periodically requesting this information from a human observer (what would be obviously an extreme case). This example shows that a context observer instance is strongly coupled to a context information instance, which characterizes a specific entity (thermometer), represented by an associated shadow object, with respect to a specifc aspect (temperature).

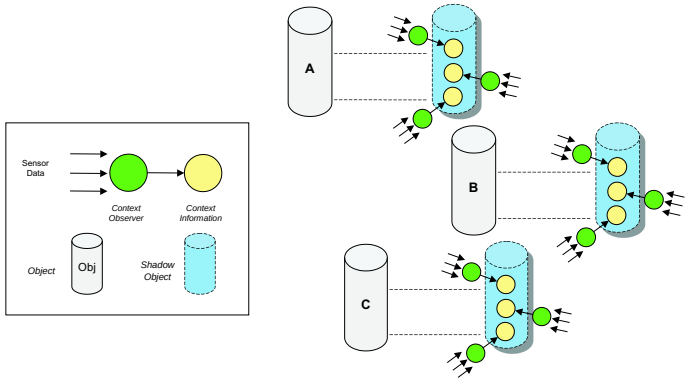

Fig. 4
ASSOCIATION OF CONTEXT INFORMATION VIA SHADOW OBJEKTS

If an entity is already a network accessible object (e.g. a mobile phone), access to the object's attributes like its call-state (on-hook or off-hook) is given by the object's interface and service endpoint. By accordingly integrating the procedure into the service platform (for instance by setting up an intermediate SOAP node [24] with a router like acting role), access to context information may be given as like as to additional attributes of the real object, without the requirement to adapt the object itself [25].

The separation between context information and its use in the service environment in the sense of the *observer pattern* [26] eases the integration of Ubiquitous Computing systems which have become very popular at the beginning of the new millenium [27]. The Ubiquitous Computing paradigm represents basically three new facets of using distributed systems:

1) Smart Devices
2) Spontaneous or Ad-Hoc Networking
3) Context Aware Services

Here the context level is not only important for the last item. Also in the area of spontaneous networking there is a strong benefit from being able to specify which aspects of the current network situation are relevant and thus must be taken into account in the sense of ad-hoc configurations [28]. Smart devices, especially small mobile connected devices like mobile phones, personal digital assistants (PDAs) or simple sensor modules (e.g. thermometers, noise sensors etc.) are on the one hand able to provide important context information about relevant entities (e.g. persons, rooms etc.). On the other hand, in particular small devices with their restricted input and output facilities as well as their resource limitations benefit from information gained from the environment of usage (e.g. the current geographic position), because this data is not required to be entered by the user. The context level uses context information which is acquired from sensors in Ubiquitous Computing environments, and which characterize the state of relevant entities.

An interesting piece of context information is particularly obtained from the mobility of the devices in Ubiquitous Computing systems: The spatial position in relation to a given reference system (e.g. WGS84). By mapping to symbolic information ("building A, room 008") and linking to relational information ("in the vicinity to") a position information becomes a

location. Services accessing a context information representing the current location are titled *location based services (LBS)*. But there is more to context than location [29]. Another very important kind of context information is obtained from time. Obviously a service may deliver very different results depending on execution before, during or after a certain point or period in time. Furthermore any information acquired by a sensor (e.g. position, time, temperature, intensity of light, speed etc.) may serve as context information. Within the so called primary context information, access is given to the raw data of the respective reference system (WGS84 coordinate, universal time, temperature in degrees Celsius, intensity of light in Lux, speed in km/h etc.). In contrast the raw data is processed and cultivated in one or more steps before representing secondary context information. From this consideration it becomes obvious, that the value range of an aspect of a primary context information always is the reference system itself. Compared to this, the definition of a classification, symbol- or value range for an aspect is coercively required to designate a closed set of possible states of a secondary context information. An example: The set

$$\mathcal{W}_{TempCelsius} = \{t | t \in \mathcal{R}, t > -273\}$$

defines the value range (set of possible states) of a primary context information representing the state of a temperature sensor. In contrast, the set

$$\mathcal{W}_{TempFeeling} = \{warm, cold\}$$

defines the value range of a secondary context information representing the state of another temperature aspect. Indeed it is up to the responsible context observer to decide which state the context information represents, for instance by performing a threshold decision. A context sensor can be seen as a function f, fusing data from n sources and assigning a valid state from the value range $\mathcal{W}$ of a certain aspect to a specific instance of a context information:

$$f(sensor_i | 1 \leq i \leq n) : \mathcal{D} \rightarrow (\mathcal{W}, p) \;\; with\, p \in \mathcal{R}, 0 < p \leq 1$$

Here p is the degree of certainty a context sensor assures from itself to be able to assign the correct state of that context information (*soft determination*).

It is important to specify within the definition of an aspect which state is caused by what conditions, but neither, how the necessary information are obtained (i.e. which sensors are accessed, definition range $\mathcal{D}$), nor how the functions f works.

A subclass of context information are the so called *highlevel quality of service (HL-QoS)*. Although the naming may be somewhat misleading, HL-QoS and their integrative potential in service environments are dealt with in several publications (i.a. in [30], [31], [32]). Highlevel QoS are in general non-functional parameters characterizing the performance, reliability, availablility or security of a service. Based on these parameters it is possible for instance within a *Web Service Level Agreement (WSLA)* [33] to specify which action in case of over- or undershooting a boundary value agreed on in a WSLA document has to be taken by whom. The language elements used are partly very well capable of defining aspects and relevance conditions. But the overall concept of WSLA is unfortunately inappropriate for our purposes because of the static composition of such a "contract" in the form of a single WSLA document, as well as the limited validity between exactly two contract signers. It is required to have a quite more flexible way of spontaneous composition and examination of the current situation, i.a. based on context information, whose existence is unknown at the time of creation of a WSLA document.

All non-functional parameters like the quality properties mentioned above are not assigned to one of the classic service interoperability levels uniformly in literature. Mani und Nagarajan [31] see them for instance as an extension of an interface specification and thus assign them to the signature level. Weller [30] proposes to access quality, payment and deadlock parameters in parallel to a service's execution, settling them at the protocol level. But most often non-functional or non-operational specifications are handled (if at all) at the semantic level. Ankolekar et. al. [13] for instance defined some few *functional attributes* named characteristics like *geographicRadius* or *degreeOfQuality*. According to their proposition a service's non-functional behaviour and the conditions to run a service can be described with these kind of attributes. In our opinion, it makes much more sense to deal with the interoperability of services with respect to contextual conditions and dependencies on a new level, the *context level*. In doing so, this enables a thematic focussing and eases the separation of services, which are interoperable with respect to the three classic levels, but not on the context level, or vice versa. An example for this are the prime services [34] of the new european satellite navigation system Galileo [35]. Although the software of a Galileo receiver is interoperable to the Galileo system in the sense of the three classic levels of service interoperability, it is interoperable to the prime services (higher degree of accuracy etc.) only in a "commercial" context (that means by paying a fee), but not in a "free" context. Another example is an electronic public transport timetable service: A client may be interoperable with such a service in city A on the context level, but not with the same service provided in city B when currently being in A, even if interoperability is given with respect to the three classic levels in both cities.

Before and during the usage of a service the relevance of a context information is very important. The set of relevant entities and the resulting context information characterizing those entities is changing rapidly. In contrast, the set of the specific aspects determining the relevance of an entity is fixed. Thus it is the aspects which have to be specified on the context level for each service. On the basis of the specified aspects it can be evaluated in advance and during execution if there are relevant entities for a service. By using the associated context information found in that way it can be decided whether the execution of a service is (from the contextual point of view) possible and advisable, or not. The set of relevant entities has influence on the service in multiple ways also at runtime. This set determines for instance if the conditions having been responsible for selecting this service during discovery are still given and valid during execution. If not, the execution of that service will be temporarily interrupted or permanently aborted. One example for this may be the physical distance to a point-of-interest

(POI): With the distance to that POI increasing, the execution of any POI specific service makes even less sense.

In the style of a *context ontolgy* introduced by Ötztürk and Aamodt [36] we call this kind of external context information a *environment related context information*, because the contextual "environment" of a service has influence on its execution. In contrast, there are the (from the service's point of view) external context information, which are target-oriented for the service itself, which is why we call them *target related context information*. An example may be a routing service from location A to location B, where A is assigned the current position of the user's device automatically by the system, obtained from the current situation. Nevertheless, we do not share Ötztürk's and Aamodt's opinion that this kind of external context information has to be seen as static during service execution time, as one can see from an example for target related context information: Because of the mobility of a user, the current position keeps changing over time. An example for an *internal context information* may be the starting time of a meeting at location B, which is used internally by the mentioned routing service for calculating the route. An overview of this classification is shown in Fig. 5.

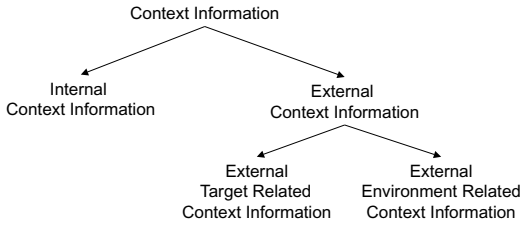

Fig. 5
CLASSIFICATION OF CONTEXT INFORMATION

We are currently defining a *Context Ontology Language (CoOL)* based on XML which may be used to exactly specify the relevant aspects of a service. See Fig. 3 on page 3 for an example of an instance document using parts of CoOL. One advantage of having such a language is to be able to describe not only the kind of aspect in a human and machine readable and evaluable way, but also to specify all possible states a context information may represent with respect to that aspect. Thus a CoOL document may contain one or more *closed* definitions of possible states of an aspect. Furthermore, by applying several different CoOL documents for the same entity, a very high flexibility and extendability can be reached. CoOL links the object-associated context information with the services itself, and thus enables context aware service usage, and additionally enables the determination of contextual service interoperability.

IV. CONTEXTUAL SERVICE INTEROPERABILITY

One of the most central questions in our current work is what are the conditions for two services and/or a service and the platform for being interoperable with respect to the unstable situation.

Vallecillo, Hernández und Troya identified in [10] two properties, *compatibility* and *substitutability*, which must be given for any interacting components to be interoperable with respect to the classical three levels of service interoperability.

Transferred to the context level, *compatibility* could be defined in a sense that two interacting components work together in a correct manner, if both sides have a common understanding of all relevant aspects. This includes the kind of selection of the relevant entities as well as the set of states a context information may represent in the relevant aspects (including the usage of the same units). This may be reached my referring to the same CoOL specification.

On the same level *substitutability* of a service by another one may be defined as having the identical set of relevant aspects. If the competence of a service for a specific task has been determined based on a certain *environmental related context information*, this service may be substituted by another one without violating contextual service interoperability only if the other service is competent for the same *environmental related context information*. An example are two services A and B, both being able to return the address of a pharmacy with emergency service in the vicinity of a given position information. Service A may be substituted in the sense of contextual service interoperability by service B if and only if B is capable of returning a valid address for any position information where A is capable of returning a valid result (we call this the same *spatial competence* of A and B). Note that the results are not necessarily the same. Likewise this does apply also with the *target related context information*: Substitutability of A by B means here that B is target-oriented in the same context like A.

In [37] a two-step-procedure is proposed to certify any kind of interoperability, which is for instance also employed in [33]: In the first step, called the logical certification, any specification of the components are checked regarding consistency, integrity etc. In the second step, called the physical certification, it is checked if an implementation follows a given specification. Whereas the logical certification is usually done by human interaction, the physical certification can be done very well by simulation or integration into real systems on trial in many cases.

An analogous procedure is recommended for checking for interoperability on the context level. After first reviewing the definitions of the aspects and any related relevance conditions given in one or more CoOL documents with respect to consistency, integrity etc. by locigal certification, the physical certification may be done by some automatism integrated into the service platform much more easily.

V. CONCLUSION AND OUTLOOK

We showed in the previous sections, that it is valuable to consider the context of a service from several perspectives, which is done insufficiently in existing research work. By introducing the new context level as another sublevel of service interoperability, we have been able to separate services, which are interoperable with respect to the three classic levels of service interoperability (signature, protocol and semantic level), but not on the context level and vice versa. We motivated the necessity of having a context ontology language. One of the objectives of combining the object-associated context information with the outlined XML based Context Ontology Language (CoOL) is to

have a tool which allows one to improve scenarios of context aware service usage. This is particularly important in modern Ubiquitous Computing environments, where we want to get the most benefit out of the huge amount of multi-networked smart mobile devices. By examination of service interoperability on the proposed context level it can be decided with the help of the tool what the contextual conditions are to be interoperable. Based on that, service handover on higher levels as well as concepts of (at least partial) autonomy may be realized.

REFERENCES

[1] A. S. Tanenbaum, *Distributed Systems*. Prentice Hall, 2002.

[2] P. Wegner, "Interoperability," *ACM computing surveys*, no. 28, pp. 285–287, 1996.

[3] S. Heiler, "Semantic Interoperability," *ACM computing surveys*, no. 27, pp. 271–273, 1995.

[4] T. Murer, D. Scherer, and A. Wuertz, "Improving component interoperability information," in *Proceedings of Workshop on Component-Oriented Programming (WCOP'96) at 10th European Conference on Object-Oriented Programming (ECOOP'96)*, pp. 150–158, dpunkt, July 1996.

[5] A. Vallecillo, J. Hernandez, and J. M. Troya, "Woi'00: New issues in object interoperability," in *LNCS 1964: ECOOP'2000 Workshop Reader*, pp. 256–269, Springer, 2000.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL)." http://www.w3.org/TR/wsdl, 2001.

[7] D. M. Yellin and R. Strom, "Protocol specifications and component adaptors," *ACM Trans. Prog. Lang. Syst.*, no. 19, pp. 292–333, 1997.

[8] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek, "Web Service Choreography Interface (WSCI)." http://www.w3.org/TR/wsci, 2002.

[9] P. Miller, "Interoperability: What is it and why should i want it?." http://www.ariadne.ac.uk/issue24/interoperability, 2000.

[10] A. Vallecillo, J. Hernández, and J. M. Troya, "Component interoperability," Tech. Rep. ITI-2000-37, Departmento de Lenguajes y Ciencias de la Computación, University of Málaga, July 2000. Available at http://www.lcc.uma.es/∼av/Publicaciones/00/Interoperability.pdf.

[11] M. Uschold and M. Grüninger, "Ontologies: Principles, methods, and applications," *Knowledge Engineering Review*, vol. 11, no. 2, pp. 93–155, 1996.

[12] J. Hendler and D. L. McGuinnes, "Darpa agent markup language," *IEEE Intelligent Systems*, vol. 15, no. 6, pp. 72–73, 2001.

[13] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng, "Daml-s: Semantic markup for web services," in *Proceedings of the International Semantic Web Workshop*, 2001.

[14] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *First Int. Semantic Web Conf.*, 2002.

[15] "Semantic Web." http://www.semanticweb.org.

[16] B. Liskov and J. Wing, "A behavioural notion of subtyping," *ACM Trans. Prog. Lang. Syst.*, no. 16, pp. 1811–1841, 1994.

[17] P. Kähkipuro, L. Marttinen, and L. Kutvonen, "Reaching interoperability through ODP type framework," Tech. Rep. C-1996-96, Department of Computer Science, University of Helsinki, June 1996.

[18] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing, Special issue on Situated Interaction and Ubiquitous Computing*, vol. 5, no. 1, 2001.

[19] A. Schmidt and K. V. Laerhoven, "How to build smart appliances," *IEEE Personal Communications*, August 2001.

[20] W. N. Schilit, *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995.

[21] M. Angermann, J. Kammann, P. Robertson, A. Steingass, and T. Strang, "Software representation for heterogeneous location data sources within a probabilistic framework," in *Proceedings of International Symposium on Location Based Services for Cellular Users (Locellus 2001)*, (Munich, Germany), pp. 107–118, February 2001.

[22] M. Banatre and P. Couderc, "Unleashing context-aware application with spatial programming," in *Keynote on IST Mobile Venue 2002: Radio Resource Management and Mobile Location Workshop*, (Athens, Greece), May 2002.

[23] P. Couderc, *Mobilité contextuelle dans les systèmes d'information*. PhD thesis, Université de Rennes-1, 2001.

[24] M. Gudgin, M. Hadley, J.-J. Moreau, and H. Frystyk, "SOAP messaging framework." http://www.w3.org/TR/soap12-part1, October 2001.

[25] M. Samulowitz and C. Linnhoff-Popien, "Interaction patterns for dynamic personalisation of service endpoints in ubiquitous computing environments," in *Accepted for KIVS 2003*, (Leipzig, Germany), 2003.

[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison-Wesley, 2002.

[27] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal Communications*, pp. 10–17, August 2001.

[28] M. Angermann and J. Kammann, "Cost metrics for decision problems in wireless ad hoc networking," in *Proceedings IEEE CAS 2002*, (Pasadena, USA), September 2002.

[29] A. Schmidt, M. Beigl, and H.-W. Gellersen, "There is more to context than location," *Computers and Graphics*, vol. 23, no. 6, pp. 893–901, 1999.

[30] S. Weller, "Online article: Web services qualification," 2002. Available at http://www-106.ibm.com/developerworks/webservices/library/ws-qual/?dwzone=webservices.

[31] A. Mani and A. Nagarajan, "Online article: Understanding quality of service for web services," 2002. Available at http://www-106.ibm.com/developerworks/library/ws-quality.html.

[32] M. F. Bertoa and A. Vallecillo, "Quality attributes for cots components," in *Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002)*, (Malaga, Spain), June 2002.

[33] H. Ludwig, A. Dan, R. Franck, A. Keller, and R. King, "Web Service Level Agreement (WSLA)," in *IBM WebService Toolkit Documentation*, 2002.

[34] V. Liebig, "Galileo," *Ignition*, pp. 2–7, September 2002.

[35] R. Schneiderman, "Position - location - whose technology?," *Wireless Systems Design*, pp. 14–20, March 2000.

[36] P. Ötztürk and A. Aamodt, "Towards a model of context for case-based diagnostic problem solving," in *Context-97; Proceedings of the interdisciplinary conference on modeling and using context*, (Rio de Janeiro), pp. 198–208, February 1997.

[37] T. D. Group, "Online article: Interoperability." Available at http://www.lsi.us.es/∼tdg/doc/topics/interoperability.html.