# CoOL: A Context Ontology Language to enable Contextual Interoperability

Thomas Strang[1], Claudia Linnhoff-Popien[2], and Korbinian Frank[2]

[1] German Aerospace Center (DLR), Oberpfaffenhofen, Germany
thomas.strang@dlr.de
[2] Ludwig-Maximilians-University (LMU), Munich, Germany
{linnhoff|frank}@informatik.uni-muenchen.de

**Abstract.** This paper describes a context modelling approach using *ontologies* as a formal fundament. We introduce our *Aspect-Scale-Context (ASC) model* and show how it is related to some other models. A *Context Ontology Language (CoOL)* is derived from the model, which may be used to enable context-awareness and contextual interoperability during service discovery and execution in a proposed distributed system architecture. A core component of this architecture is a reasoner which infers conclusions about the context based on an ontology built with CoOL.

## 1 Introduction

The trend towards pervasive computing [1] is driving a need for services and service architectures that are aware of the *context* of the different actors (any user, any service provider and even the environment or third parties) involved in a service interaction. For instance, context information (for definition of terminology see section 2) can be used to reduce the amount of required user interaction, as well as to improve the user interface of small mobile devices such as mobile phones [2], which are typical for pervasive computing scenarios. A key accessor to context information in any context-aware system is a well designed model to describe contextual facts and contextual interrelationships. Several approaches from the early days of modelling the context typically lack formality and are primarily concerned with requirements for the model from the customer perspective. More recent proposals such as [3] try to countersteer the lack of formality by introducing a graphical oriented approach to model contextual interrelationships. The context modelling approach introduced in this paper tries to close the formality gap by using *ontologies* [4] as a fundament to describe contextual facts and interrelationships. Particularly, this allows to determine service interoperability on the context level [5].

This paper is organized as follows: In section 2 we will introduce our ASC model after giving a motivation why we make use of ontologies as a fundament of our model. Section 3 shows how our ASC model can be used as transfer model for other proposed context models, considering a graphical context model as an example. In section 4 we propose a way how to plug in our ASC model

J.-B. Stefani, I. Demeure, and D. Hagimont (Eds.): DAIS 2003, LNCS 2893, pp. 236–247, 2003.

into DAML-S. A context extension of a well established general purpose service model shown in section 5 motivates the design of our system architecture in section 6. Because relevance is more than just spatial and temporal proximity, we describe our approach of expressing relevance criteria in section 7, before we summarize our paper with a conclusion in section 8.

## 2 Model

Because of the fact that the terms *context* etc. in current publications are used in various ways it is necessary to define the terminology we use. The following is a short reflection of our terminlogy used throughout this paper, a more comprehensive introduction to this terminology can be found in [5]: 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, grouped in one or more related dimensions called *scales*. 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 concerning 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. The *situation* is the set of all known context information. These definitions are very similar to other definitions of context (e.g. [6,7,8]), but refine the expressiveness by introducing the terminology of an *aspect*, which is discussed in detail in section 2.2.

### 2.1 Ontologies and the Context Ontology Language

When dealing with context information it is always a challenge to describe contextual facts and interrelationships in a *precise* and *traceable* manner. For instance, to perform the task "print document on printer near to me", it is required to have a precise definition of terms used in the task, particularly what "near" means to "me". It is highly desirable, that each participating party in a service interaction shares the same interpretation of the data exchanged and the meaning "behind" it (so called *shared understanding*). This is done in our approach by the use of *ontologies* [4]. Ontologies seem to be well suited to store the knowledge concerning context.

An ontology is a specification of a conceptualization [9]. The term "ontology" itself is borrowed from philosophy, where it has a long history in refering to the subject of existence. In IT systems, ontologies are used to express the more or less complete knowledge about concepts (classes of subjects) and their attributes, as well as their interrelationships. Ontologies may be stored at different places and created by different authors, which offers the amount of flexibility and extensibility we need in distributed systems. The merging of different ontology fragments is one of the main tasks of a *reasoner*, which is called *inference*

*engine* if it infers knowledge from symbolically coded axioms. A reasoner may be queried via some query language to deliver instances and their values, as well as concept and attribute names based on the ontologies known to the reasoner. A reasoner may also be used to validate consistency (within one ontology, but also with respect to related ones), and to assert inter-ontology relationships and "complete" the ontologies by computing implicit hierarchies and relationships based on given rules. One of the most advanced inference engines is the *OntoBroker* system [10], which we used to evaluate most of the inferencing issues during our research.

There have been proposed several languages designed for or being even able to describe ontologies in recent years. We analysed several ontology specification and query languages with a focus of the following two questions:

1. How well is the language capable to describe concepts, attributes and relations in a precise and traceable manner? (knowledge representation)
2. How well is the language capable to create effective queries towards the reasoner? (knowledge querying)

We found out, that any of the analysed languages which has advantages w.r.t. the first question has disadvantages w.r.t. the second one and vice versa. With this background we defined our *Context Ontology Language (CoOL)* not as a single, monolithic language. Instead, it is a collection of several fragments, grouped into two subsets.

The first subset, *CoOL Core*, is a projection of our model, which will be introduced in section 2.2, into two (three) different common ontology languages:

- OWL and DAML+OIL, which are both part of the Semantic Web's [11] ontology languages based on XML and RDF/S. See figure 2 on how *CoOL Core* is related to the Semantic Web stack. Because OWL is the successor of DAML+OIL covering nearly the same issues, we will use the term OWL as a representative for both languages in the reminder of this paper unless stated otherwise.
- F-Logic [12], which is a logic language combining object-oriented and predicate logic characteristics not based on XML.

The second subset, *CoOL Integration*, is a collection of schema and protocol extensions as well as common subconcepts of the model introduced in the next section, enabling the use of *CoOL Core* in several service frameworks, particulary Web Services. This paper deals mainly with *CoOL Core* and the model it is based on, whereas *CoOL Integration* is somewhat out of the focus of this paper.

Having a projection of the model in multiple ontology languages enables the following proposed procedure: For the knowledge representation issues, a developer may use any of the languages which seems to be adequate, e.g. using OWL because of the wide range of available tools helping to create and/or validate ontology fragments, or using F-Logic because of its object-oriented, compact syntax or its rule based extensibility. For knowledge querying issues, we preferred to use the *OntoBroker* inference engine, most notably because it supports F-Logic as

knowledge representation *and* knowledge query language. This decision requires any knowledge not represented in F-Logic to be converted into F-Logic first, but this is no real disadvantage, because most other major reasoners have a similar requirement. This conversion is possible as long as the features of OWL in use do not exceed a certain subset (OWL-DL), as Borgida showed in [13]. F-Logic is more expressive than OWL, can be used as query language as well, and is much more appropriate for specifying *relevance conditions* (see section 7).

### 2.2 Our ASC Model

Our *Aspect-Scale-Context (ASC)* model is named after the core concepts of the model, which are *aspect*, *scale* and *context information*, see figure 1. Each *aspect* aggregates one or more *scales*, and each *scale* aggregates one or more *context information*. These core concepts are interrelated via *hasAspect*, *hasScale* and *constructedBy* relations.

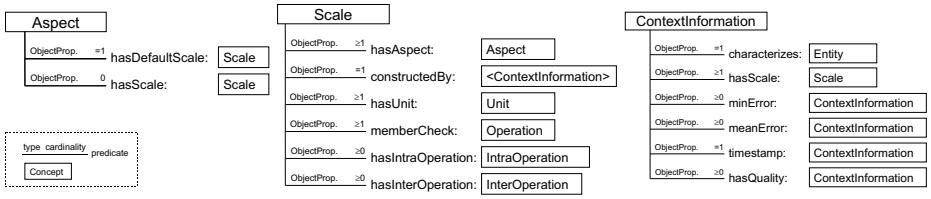


**Fig. 1.** Aspect-Scale-Context (ASC) Model

As anchored in the definitions at the beginning of the section, an aspect is a set of one or more related scales. Likewise, any aspect is a dimension of the situation space, being used as a collective term for information objects having the same semantic type.

A scale is an unordered set of objects defining the range of valid context information. In other words, a valid context information with respect to an aspect is one of the elements of the aspect's scales. For instance the aspect "GeographicCoordinateAspect" may have two scales, "WGS84Scale" and "GaussKruegerScale", and a valid context information may be an object instance created in an object oriented programming language such as Java with `new GaussKruegerCoordinate("367032", "533074")`. Scales based on primitive datatypes such as scalars instead of objects are captured by corresponding wrapper classes. Thus a valid context information of the aspect "SpatialDistanceAspect" with the given scales "MeterScale" and "KilometerScale" may be an object instance created with `new Integer(10)`.

On an abstract level, context information may be seen as content data complemented by some meta data characterizing the content data. Each context information has an associated scale defining the range of valid instances of that type of context information. Context information characterizing the content of another context information is a meta information and thus a context information of higher order and expresses the quality of the lower order context

information, see figure 3. *CoOL Integration* includes already a set of standard quality aspects such as a *minimumError*, a *meanError* and or a *timestamp*, but any other kind of context information characterizing the quality of another context information may be assigned to the context information of interest using the *hasQuality* property of the ASC model.

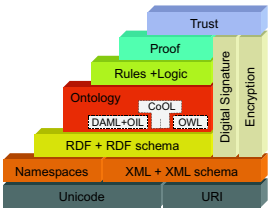


**Fig. 2.** CoOL embedded inside the Semantic Web stack

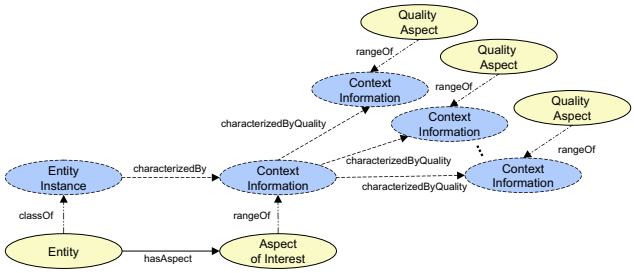


**Fig. 3.** Context Information being an Entity itself

As mentioned, scales are sets of context information. Each scale is constructed by one class of context information such as "WGS84Coordinate" and "Gauss-KruegerCoordinate" in the example of the last subsection. All scales within one aspect are constrained by the ASC model in a way, that there must exist a mapping function from one scale to at least one other of the already existing scales of the same aspect. This function is called *IntraOperation*, see figure 4.

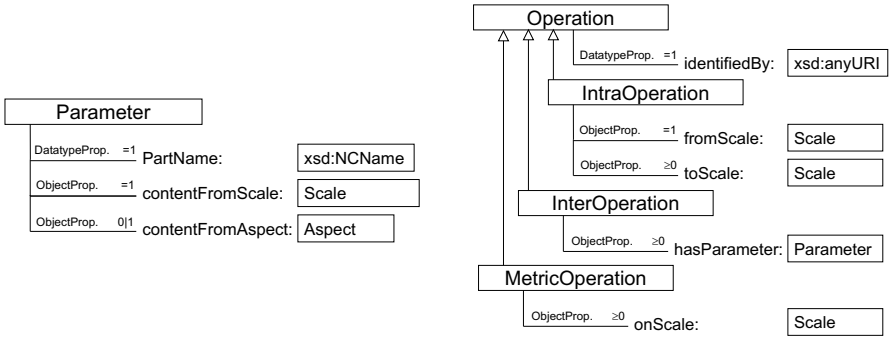


**Fig. 4.** Operations

Like that, it is possible to access every scale from every other scale of the same aspect by a series of IntraOperations. In other words, a new scale of an aspect may be constructed by providing an IntraOperation from an existing scale. This allows to build multiple related scales by providing different IntraOperations representing different scaling factors ("nautical miles", "km" or "m" for a "SpatialDistanceAspect" aspect). Depending on where the IntraOperation is specified (at the source scale or at the destination scale), the corresponding property *toScale* or *fromScale* has to be set.

Scales which require access to scales of one or more other aspects can be defined using *InterOperations*, see also figure 4. An example for such a scale would be "KilometerPerHourScale" of a "SpeedAspect" aspect. This scale can be defined using an InterOperation with two *Parameter*, delta_s and delta_t, where the parameter delta_s is from an aspect "SpatialDistanceAspect" and delta_t is from an aspect "DurationAspect".

Due to the fact that each scale is an *unordered set* of context information instance objects, there may be no relative sort order between the context information inherently given. Therefore we introduced the *MetricOperation* which may be used to compare two context information instance objects of the same scale in an implementation-defined manner to see if they match or what their relative sort order is by returning either the first or the second parameter. Thus the return value indicates the ordering of the two objects.

Information about the signature of any InterOperation, IntraOperation or MetricOperation is available in the signature specification pointed to with the property *identifiedBy*, e.g. an operation within a WSDL file or an AtomicProcess within a DAML-S grounding [14].

## 3 Transfer Model

The ASC model may be used as transfer model to employ the knowledge expressed in other context models. A good example is the nicely designed graphics oriented context model introduced in [3] by Henricksen et al., which is a context extension to the Object-Role Modelling (ORM) approach. In ORM, the basic modelling concept is the *fact*, and the modelling of a domain using ORM involves identifying appropriate fact types and the roles that entity types play in these.

Henricksen extended ORM to allow fact types to be categorised, according to their persistence and source, either as *static* (facts that remain unchanged as long as the entities they describe persist) or as *dynamic*. The latter ones are further distinguished depending on the source of the facts as either *profiled*, *sensed* or *derived* types. Using our ASC model, facts can be modelled as context information. In doing so, Henricksen's classification can be mapped by introducing a quality aspect consisting of the scale with the elements {static, dynamic profiled, dynamic sensed and dynamic derived}, which may be used to characterize any context information in a quality sense. Henricksen's quality indicators may be directly mapped to some quality aspect, similar to her history fact types, which may be addressed with a timestamp and/or time period quality aspect, which are both already basic aspects in CoOL. The last extension to ORM made by Henricksen for context modelling purposes are fact dependencies, which represent a special type of relationship between facts, where a change in one fact leads automatically to a change in another fact: the *dependsOn* relation. This behaviour is expressed in the ASC model by the existence of one or more corresponding Intra/InterOperations between the scales a pair of context information is based on each. Here our model is even more expressive, because it allows to specify exactly the kind of dependency. This example shows the potential of the ASC model to be used as transfer model for other context model approaches.

## 4 Relation to DAML-S

In the framework of the Semantic Web there has been done some serious effort in designing technologies that allow to discover, invoke, compose and monitor web resources. Among them there has been created an *ontology of services* called *DAML-S* [14], which can be used to create computer-interpretable descriptions of services from multiple perspectives. Within DAML-S three essential types of knowledge about a service have been identified: *ServiceProfile*, *ServiceModel* and *ServiceGrounding*.

Some selected elements of the current version of DAML-S find a corresponding counterpart in our Context Ontology Language. For instance, the non-functional attribute *geographicRadius* of a DAML-S ServiceProfile may be expressed as a context information based on an aspect *scope*, which is one of the default aspects within CoOL, whereas the non-functional attribute *qualityRating* may be mapped to some quality aspect. DAML-S covers only a few contextual aspects, and their specification is not very formal. To have a much more formal and thus computer-interpretable approach to describe the contextual requirements and impact of a service, we suppose to extend DAML-S with a fourth type of knowledge about a service, dealing with the contextual issues.

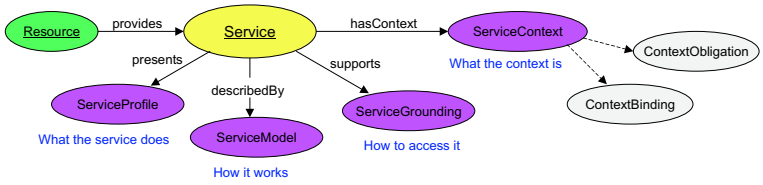


**Fig. 5.** DAML-S with Context Extension

This new perspective (we call it *ServiceContext*, see figure 5) may serve as a more formal description of a service's contextual interoperability by providing a comprehensive but extensible model based on the ASC model. The obligations of a service w.r.t. the context of its usage (e.g. the geographic scope "delivery area" covered by the service with respect to a well defined aspect "region") can be expressed in a *ContextObligation* submodel. Another submodel *ContextBinding* may be used to establish a virtual link from some input or output parameter of an *AtomicProcess* of a *ServiceGrounding* to a specific aspect, enabling automatic determination of valid or even optimal parameters.

## 5 MNM Service Model and the Context Extension

The Munich Network Management (MNM) team introduced in [15] a generic model of commonly needed service-related terms, concepts and structuring rules to describe a service from different perspectives (e.g. service view vs. implementation view). Their model is intended to analyze, identify and structure the

necessary actors and the corresponding inter- and intra-organizational associations between these actors. In this model the actors are grouped in either the *customer domain*, or the *service provider domain*. Structural elements which cannot be associated to any of these two domains are called *side independent*. In the model's *service view* these elements "build" the service in an abstract manner, thus prefer to call this set of elements the *abstract service*.

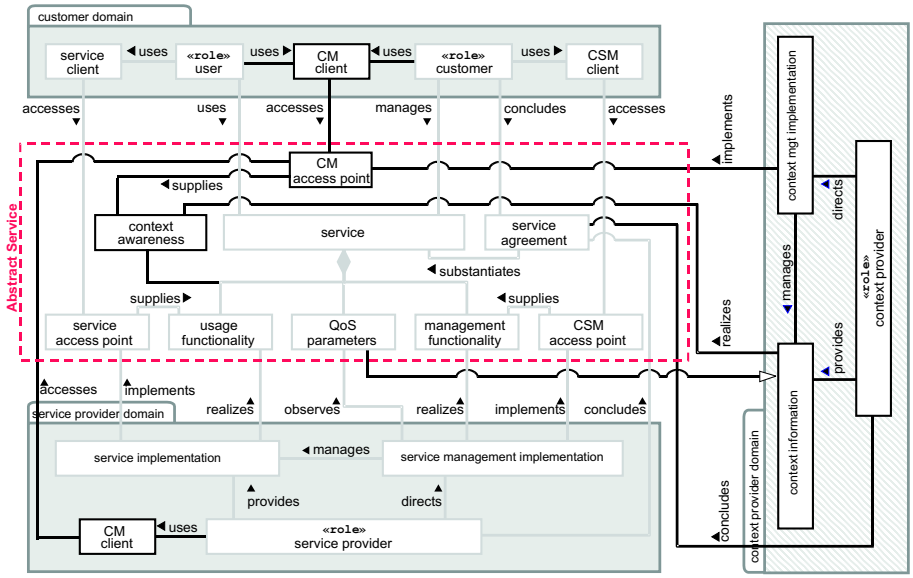


**Fig. 6.** Service Model with emphazised Context Extension (MNMplusCE)

The MNM service model has been designed primarily with network management tasks and carrier services in mind. But due to the level of abstraction this model can be applied to highlevel (non-carrier) services perfectly. Moreover, the model fits direct service usage approaches (client ↔ server) as well as intermediate service usage approaches (client ↔ middleware ↔ server). In the latter one a middleware component "fulfills" an abstract service, i.e. it behaves like a client towards the service provider domain, and behaves like a service provider towards the service customer domain by providing at least a service access point.

To be able to describe contextual dependencies and issues of context provisioning with the model in a similiar and consistent way, we extended the MNM service model with a *context provider domain* (see figure 6 on the right). This domain groups the actors responsible to manage the context observation, context processing and delivery as context information.

The context provider is not yet another service provider. Its extraordinary position is caused by, among others, the fact of being involved as a third party service provider in an interaction between the customer domain and a set of service provider domains simultaneously. The context management implementation

offers a context management access point (CMAP) to give access to the context to both other domains in the model (customer domain and service provider domain) to enable context-aware services and context-aware service usage. A service client, a service provider or even a middleware component may determine the entities relevant for a specific task by an interface provided by the CMAP, which causes the associated context management implementation to deliver context information in an adequate manner.

The context management implementation offers a *context management access point (CMAP)* to give access to the context to both other domains in the model (customer domain and service provider domain) to enable context-aware services and context-aware service usage. A service client, a service provider or even a middleware component may determine the entities relevant for a specific task through an interface provided by the CMAP, which causes the associated context management implementation to deliver context information in an adequate manner. In the latter we refer to this contextual extended MNM service model as *MNMplusCE service model*. In this model, carrier services are a specialized derivative of highlevel services, and the Quality-of-Service (QoS) parameters describing a service instance are derived from context information in the model.

## 6 System Architecture

The overall architecture of our system is shown in figure 7, with a focus on the context provider domain as introduced in the preceeding section.

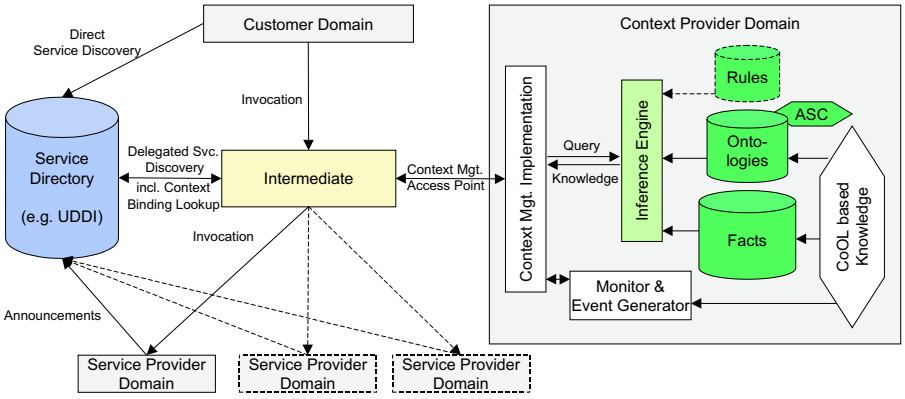


**Fig. 7.** System Architecture

This architecture enables the Intermediate as a middleware component to "fulfill" an abstract service (see section 5), in particular it behaves like a client towards the service provider domain, and behaves like a service provider towards the service customer domain. By resolving the binding between the parameters of a service call and information from a context provider (*context binding*) at

runtime, the intermediate is a central component in our architecture to enable context-awareness.

The context management implementation inside the context provider domain implements the context management access point (CMAP) interface and is responsible for the mapping to the query language (e.g. F-Logic) used to query the inference engine (e.g. OntoBroker). This engine is feeded with knowledge from different sources, specifying knowledge as conclusions from ontologies and facts based on those ontologies. Due to the fact that our ASC model is one of the base ontologies, the inference engine is able to determine knowledge about entities, aspects, scales and context information as desired for our purpose.

Any party interested in asynchronous notification about specific context conditions ("notify me when I am near the restrooms") may register with the context provider with a corresponding condition statement via the CMAP. The context provider is responsible for re-checking these conditions each time a part of the condition statement is affected by a change in the knowledge base.

## 7 Relevance

As mentioned we consider an entity as relevant for a specific task, if its state is characterized at least concerning one relevant aspect. Consequently we considered an aspect as 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. The separation between customer domain, service provider domain and context provider domain in the MNMplusCE service model (see section 5) makes it necessary to declare the task-specific relevance towards the task-independent context provider domain through the CMAP. In other words, by specifying relevance conditions an abstract context-aware service becomes a concrete context-aware service.

We distinguish between external and internal relevance, depending on the domain of relevance determination. If the relevance of an entity is identified outside the context provider domain, e.g. at the customer domain, we call this an external relevance. In this case the other domain advises the context provider domain through the CMAP to a specific entity by an entity identifier, and the context provider delivers context information assigned to that entity. An example for an external relevance is a service client identifying an entity representing its own device. By sending this entity representative to the context provider domain, the context management implementation may be able to determine the current position of the client device by some sensor, and deliver an adequate context information to the service client, which uses this information while invoking the service implementation.

In contrast, if the relevance of an entity is identified in the context provider domain itself, we call this an internal relevance. It is essential to enable the context provider domain to determine internal relevance of one or more entities by providing some relevance condition.

A internal relevance condition is a filter, which can be used at several levels to identify one or more relevant entities out of the set of all known entities in the context provider domain. A first level filter specifies only an aspect of interest

(e.g. "get all entities where you know something about the aspect *place*"). The corresponding F-Logic query would be similar to

```
FORALL E,C,S <-
   C:"urn:cool"#ContextInformation AND C["urn:cool"#characterizes->E] AND
   C["urn:cool"#hasScale->>S] AND S["urn:cool"#hasAspect->>"urn:aspects"#Place].
```

A second level filter specifies a condition about the context information based on the aspect of interest (e.g. "get all entities where you know that the current state with respect to the aspect *place* is *near*"). The corresponding F-Logic query would be similar to

```
FORALL E,C,S,V <-
   C:"urn:cool"#ContextInformation AND C["urn:cool"#characterizes->E] AND
   C["urn:cool"#hasScale->>S] AND S["urn:cool"#hasAspect->>"urn:aspects"#Place] AND
   C["urn:cool"#hasValue->V] AND equal(V,"urn:dist"#Near).
```

A third level filter specifies a condition about the quality based on the aspect of quality of a context information based on the aspect of interest (e.g. "get all entities where you know that the current state with respect to the aspect *place* is *near* and that information with respect to the quality aspect *age of information* is *less than or equal to 10 seconds*"). The corresponding F-Logic query would be similar to

```
FORALL E,C1,C2,S1,S2,V1,V2 <-
   C1:"urn:cool"#ContextInformation AND C1["urn:cool"#characterizes->E] AND
   C1["urn:cool"#hasScale->>S] AND S1["urn:cool"#hasAspect->>"urn:aspects"#Place] AND
   C1["urn:cool"#hasValue->V1] AND equal(V1,"urn:ci"#Near) AND
   C1["urn:cool"#hasQuality->>C2] AND C2["urn:cool"#hasScale->>S2] AND
   S2["urn:cool"#hasAspect->>"urn:aspects"#Age] AND C2["urn:cool"#hasValue->V2] AND
   lessorequal(V2,10).
```

Figure 3 on page 240 also illustrates the distinction between the levels visually. Note that the context information characterising an entity w.r.t. the aspect of interest is treated as an entity instance itself when characterized w.r.t. some quality aspect. The party specifying the relevance condition may be interested in the respective context information instead of the entities characterized by these context information. Thus the context provider may offer two separate functions to distinguish between them. The modification of the F-Logic query would be as easy as deleting the variable "E" in the FORALL part and any partial term containing the "E".

## 8 Conclusion and Outlook

In the previous sections we introduced the ASC model as a base model to express how some context information can be used to characterize the state of an entity concerning a specific aspect. A high degree of formality has been reached by using ontologies as a fundament for the model, which guarantees good automatic interpretation capabilities of an implementation of the model. We showed, how the ASC model fits into a general purpose service model where we made a context extension, making any service interaction based on that model context-aware. In our proposed system architecture the ontology reasoner is employed to determine interrelationship dependencies and relevance conditions, which may affect a service interaction at any stage. It became very clear that we consider

relevance to be more than just spatial and temporal proximity. Finally we showed how our model can be used as transfer model by means of a specific example, and how it is related to DAML-S. Further work has to be done to complete the latter one when the DAML-S specification itself is officially released and stable.

## References


1. Satyanarayanan, M.: Pervasive computing: Vision and challenges. IEEE Personal Communications (2001) 10–17
2. Strang, T.: Towards autonomous context-aware services for smart mobile devices. In Chen, M.S., Chrysanthis, P.K., Sloman, M., Zaslavsky, A., eds.: LNCS 2574: Proceedings of the 4th International Conference on Mobile Data Management (MDM2003). Volume 2574 of Lecture Notes in Computer Science (LNCS)., Melbourne/Australia, Springer Verlag (2003) 279–293
3. Henricksen, K., Indulska, J., Rakotonirainy, A.: Generating context management infrastructure from high-level context models. In: Industrial Track Proceedings of the 4th International Conference on Mobile Data Management (MDM2003), Melbourne/Australia (2003) 1–6
4. Uschold, M., Grüninger, M.: Ontologies: Principles, methods, and applications. Knowledge Engineering Review **11** (1996) 93–155
5. Strang, T., Linnhoff-Popien, C.: Service interoperability on context level in ubiquitous computing environments. In: Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w), L'Aquila/Italy (2003)
6. Dey, A.K.: Understanding and using context. Personal and Ubiquitous Computing, Special issue on Situated Interaction and Ubiquitous Computing **5** (2001)
7. Schmidt, A., Laerhoven, K.V.: How to build smart appliances. IEEE Personal Communications (2001)
8. Schilit, W.N.: A System Architecture for Context-Aware Mobile Computing. PhD thesis, Columbia University (1995)
9. Gruber, T.G.: A translation approach to portable ontologies. Knowledge Acquisition **5** (1993) 199–220
10. Decker, S., Erdmann, M., Fensel, D., Studer, R.: Ontobroker: Ontology based access to distributed and semi-structured information. In et al., R.M., ed.: Semantic Issues in Multimedia Systems, Boston/USA, Kluwer Academic Publisher (1999) 351–369
11. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American **284** (2001) 34–43
12. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. ACM **42** (1995) 741–834
13. Borgida, A.: On the relative expressiveness of description logics and predicate logics. Artificial Intelligence **82** (1996) 353–367
14. Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., Zeng, H.: Daml-s: Semantic markup for web services. In: Proceedings of the International Semantic Web Workshop. (2001)
15. Garschhammer, M., Hauck, R., Kempter, B., Radisic, I., Roelle, H., Schmidt, H.: The MNM Service Model — Refined Views on Generic Service Management. Journal of Communications and Networks **3** (2001) 297–306