Applying Context-awareness to Service-oriented Architecture

Florian Kronenberg

RWTH Aachen University, Aachen, Germany
florian.kronenberg@rwth-aachen.de

Abstract. Mobile devices used today have limited capabilities of gathering information about the situation in which they are used – the context – collecting data by sensors built into the devices. The currently popular service-oriented architecture paradigm composes applications from distributed services which provide a specific function through a platform-independent interface. In this paper, we describe possible relationships between context-aware applications and service-oriented architectures. The paper takes a close look on architectures of middleware that provide context management capabilities to service-oriented environments. We further propose a layered reference architecture following commonalities between the systems we inspected. We describe several systems as examples for applications benefiting from context-awareness, such as context-sensitive service discovery.

1 Introduction

Today mobile computing devices are used by an increasing number of worldwide users. These devices are already part of the daily life of their carriers. Contrary to the limitations of their predecessors, many current mobile devices have decent computing power, a great amount of expansible memory and provide networking capabilities on local and global scales. In these environments it is possible to execute powerful mobile applications. Still limited, however, is the user interface of these devices. Mobile devices are restricted in size and therefore display and input parts cannot provide a user experience that is comparable to the user experience in desktop computers. The amount of necessary interaction with mobile applications can in some situations be reduced by exploiting contextual information about users and devices. Sensor hardware that can be used for gathering contextual information is highly available and increasingly present in mobile standard devices. The low-level sensing data, such as physical location or temperature, can be used in order to infer higher-level contextual information, such as a situation (e.g. "taking a sunbath at the beach"). The user experience in mobile applications that depend on contextual information can be greatly enhanced by automatically providing contextual information derived from sensor data to these applications.

Service-oriented architecture (SOA) is a recent computing paradigm where specialized tasks are carried out by loosely-coupled, distributed software com-

ponents which are accessible through platform-independent protocols and interfaces. Services can be invoked by or composed to various applications. The main goal of such an architecture is that services provide functions – the industry mostly considers them business processes – which need to be implemented and deployed only once in order to be used by several applications.

Both concepts, context-aware computing and the service-oriented architecture paradigm, can mutually take advantage of each other. Context-aware systems need a *context management* component in order to abstract from raw sensor data and deliver higher-level contextual information to other functional parts of the system. Mobile devices which carry sensors are usually limited to detecting only a certain part of the overall system context. The service-oriented paradigm helps in building *middleware* solutions which make certain pieces of contextual information available to components to which they are relevant, regardless of the physical distribution of the sensors from where this information originates. Access to contextual information on different levels of abstraction can be published as a service and these services can be composed to deliver higher-level contextual information – again to be published as services. On the other hand, service-oriented architectures can be enhanced by context-awareness. Some authors report benefits from using context information in order to improve the quality of search results in *service discovery*. Others propose that the *service behavior* itself could change depending on contextual information. These services can provide capabilities that adapt themselves to the specific contexts of the service clients and service providers.

This paper inspects systems which unite context-awareness and service-orientation. All of these systems provide some type of context-management, which we will inspect in more detail. Some of them use context-management in order to build context-aware middleware solutions to be used within service-oriented architectures. We present another important example on how contextual information can be used within service-oriented architectures: context-aware service discovery.

The systems we discuss in detail are a selection from scientific literature. They were chosen according to their affinity to the service-oriented paradigm and their general relevance as indicated by the number of other publications containing references to them. Selected systems providing context-management in a service-oriented architecture are:

- *ContextToolkit* [31,32]
- *Reconfigurable Context-Sensitive Middleware (RCSM)* [31,32]
- *Context-Aware Middleware for Ubiquitous Computing Systems (CAMUS)* [2,3,4,5,33]
- *Context Ontology Language (CoOL)* [2,3,4,5,33]
- *Service-Oriented Context-Aware Middleware (SOCAM)* [35,36]

Our selection of systems concerning context-aware service discovery is:

- Untitled context-aware service discovery system by Doulkeridis et al. [18]
- *Context-Sensitive Service Discovery System (CSDS)* [19,20]

The rest of the paper is organized as follows: Section 2 introduces some general concepts of context management and defines a layered reference architecture for context management applying these concepts. In section 3 we describe languages and standards that have proven useful in context-aware service-oriented architectures according to the systems we inspected. In the following section we summarize the internal architecture of the context-management systems we inspected as their role in the overall architecture in which they were proposed. We also inspect implementations of an important application of context-awareness in service-oriented architectures, context-aware service discovery. The last section defines evaluation criteria and evaluates the inspected systems according to the criteria.

2 General Concepts of Context Management

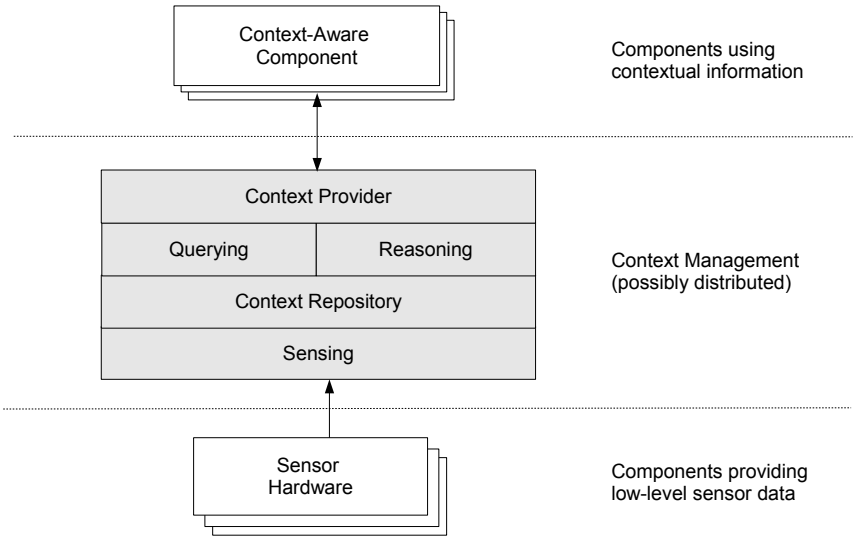
Context-aware systems manage context on different levels of abstraction [1]. Contextual information can be represented by a single scalar value (such as "room temperature is 25 degrees centigrade") which represents the raw output of a sensor. Contextual information can also be a higher-level semantic description of a situation (such as "waiting for the bus"). Sophisticated context management systems are able to deal with both types of context information, combining and transforming the former to the latter. Context information that is delivered to the system by sensing units has to be processed and transformed from raw sensor signal data to representations of the context that are structured according to a certain context model. These representations allow other components of the context-aware systems to access the context on different levels of abstraction through unified access points.

In this section we introduce concepts that are common to systems that provide context management. Based on the context-management systems we inspected and the commonalities we found between them we propose a layered reference architecture (see fig. 1). Each of the concepts is assigned to a layer according to the level of abstraction of the contextual information provided, starting with sensing at the bottom layer and ranging up to high-level context provision to collaborating systems. Access to the context data representations of each layer is granted through well-defined interfaces hiding the intrinsic details of the lower layers. Each higher layer optionally combines information gathered from multiple distributed instances of the next lower layer. The distributed instances provide different but not necessarily disjoint portions of the contextual information which forms the overall system's context.

Sensing deals with the processing and transformation of raw sensor data. The *context data repository* stores the system's context-related state in appropriate data structures. *Context reasoning* allows the semantic extraction of higher-level context information from lower-level context information while *querying* refers to methods used in order to access selected contextual information from higher layers. A *context provider* is the top layer that publishes contextual information as a service in a service-oriented architecture. Most of these concepts are present

in a majority of the systems that we inspected. We chose concepts that allow a complete model of context management from sensing to high-level context provision. Some of the systems, however, do not focus on a complete representation and did therefore not include all of the concepts. Others include all of the concepts but do not prescribe a layered architecture.

Fig. 1. A proposed layered reference architecture for context-management.



2.1 Sensing

Contextual facts can be discovered by interpreting data that is collected by sensors. Sensing units produce data tuples that represent the state of an observed entity according to certain scales. Typical examples for context information that can be observed by sensors are location (eg. via GPS), movement, temperature, time and proximity of other entities (by RFID, bluetooth, etc.).

Access to and representation of sensor data is likely to be heterogeneous in most systems. Sensing units can be connected to the system via a variety of hardware and software interfaces and the native access to these units can be tied to different layers of abstractions within the systems. The manufacturers of some sensing units, for example, can provide a high-level SDK in order to access the units while others might require developers to parse a bit stream from the serial port. This leads to increasing complexity in the design and implementation of

context-aware system. A layer of abstraction, which allows the sensor data to be accessed by higher layers in a standardized and unified manner, should be provided in order to prevent strong dependencies of a context-aware system on specific sensor hardware. Such a layer would transform the data from sensor domain to context domain, providing a mapping between the sensor representations of observed entities and their states and a representation that is meaningful in context-management.

This concept layer is described in detail for *Context Aware Middleware for Ubiquitous Computing (CAMUS)*, which introduces the concept of Feature Extraction Agents (FXA) in order to implement a Unified Access Pattern [2,3,4,5]. FXA denote software components that extract the most descriptive features from the sensors that are needed for deducing high-level context information in the upper layers. Hence, the heterogeneity of the sensor data representation becomes transparent to the upper layers.

2.2 Context Data Representation in the Context Repository

A context repository stores the context data that is acquired by the sensing layer and provides access to it to higher layers. In order to define data structures for context data that can be processed by machines we need a formal model to represent the context. Strang et al. summarize and evaluate common modeling approaches in a 2004 survey [6]. We supplement the modeling approaches by Strang with some more recent examples and one additional approach, term based models. The approaches are classified by the characteristics of the data structures that are used to exchange contextual information throughout the system:

Key-Value Models. These are the most simple data structures modeling contextual information. Each type of context information is represented as a key that is mapped to the current state of the context information. This class of models is used for service discovery in several distributed service frameworks. The service discovery procedure of these frameworks applies a matching algorithm to the respective key-value pairs of contextual information within the service description and the contextual information within a service discovery request. An example using key-value based context attributes for service discovery can be found in [7].

Markup-Scheme-Models. Markup based models use a hierarchical data structure consisting of markup tags with attributes and content. A typical example for these models are *profiles* such as the Composite Capabilities / Preference Profile (CC/PP) [8] and the User Agent Profile (UAProf) [9], which are serializable in XML and accessible by RDF/S [10]. More examples for markup based context models can be found in [6].

Graphical Models. The UML is a general-purpose graphical modeling instrument that can be extended to model context. A UML profile for Model-Driven Development (MDD) that allows developers to graphically specify the contexts that

impact an application is proposed by Ayed et al. in [11]. An extension to ORM [12] that allows ORM fact types to be categorized as static or dynamic, context-dependent, is described by Hendricksen et al. in [13]. These modelling approaches are meant rather to bring context-awareness to the software engineering process than serving as a runtime representation of context.

Object Oriented Models. The main benefits of object oriented approaches are encapsulation and reusability. The details of context processing are available to the objects representing a certain context information only and therefore hidden from other components in the system. Access to the context information of an object is provided by well-defined public interfaces. In section 4 we give a summary of the *Context Toolkit* where context is represented in an object-oriented fashion.

Logic Based Models These models define context in terms of facts, expressions and rules. Context information can be either represented as a fact, in case of lower-level context information, or inferred applying the defined rules of a system to these facts, in case of higher-level context information. Therefore, these models require a high degree of formality. In addition to modeling context in data structures logic based models inherently emphasize on context reasoning, which is not necessarily part of the context repository from an architectural point of view but usually provided by a separate subsystem in the same or even on a higher layer.

Ontology Based Models These models represent context based on *ontologies* [14]. Ontologies are a concept borrowed from philosophy. In computer science they are used to express knowledge about concepts (or classes), their attributes and interrelationships. Ontologies may be stored and maintained by different authors in different locations which makes them suitable for distributed computing. A widely adopted standard notation for defining ontologies is the *Web Ontology Language* (OWL)[15]. They are also an important concept of the *Semantic Web* [16].

In their evaluation of different context modeling approaches Strang et al. [6] arrive at the conclusion that the ontology based models are the most promising due to their high and formal expressiveness.

Strang et al. propose a context model based on ontologies known as the *Aspect-Scale-ContextInformation (ASC)* model [17]. The core concepts of this model are:

Context Information Any information characterizing the state of an *entity* w.r.t. a specific aspect.

Entity An object, such as a person, place or a device.

Aspect Classification, symbol or value range of all reachable states in one or several dimensions called *scales*.

Scale An unordered set of objects defining the range of valid context information.

An aspect in this model could be temperature or geographical location, for example. In the case of temperature this aspect could be measured in scales such as degrees Celsius or degrees Fahrenheit. Possible would also be to have a rougher scale {chilly, cold, mild, warm, hot}. The quality of a context information, such as "30 degrees Celsius" or "hot", can be further characterized by meta information, which is a context information itself, measured in a scale of a higher-order aspect, such as quality.

The scales in an aspect are further constraint in a way that there must exist a mapping function from one scale to at least one other scale of the same aspect. This function type is called *IntraOperation*. The other two function types in this model are *InterOperations*, which produce context information depending on scales of one or more other aspects, and *MetricOperations,* which compare context information instances of the same scale to each other, establishing a natural ordering of these instances.

Term-Based Models In addition to the models that were included in the 2004 survey by Strang et al. term-based models are found in more recent context-aware systems. Term-based context models are used in systems where context is needed in order to enhance the quality of the results of service discovery. Mobile applications have particular needs regarding the relevance of the top K results since their display and user input capabilities are usually not appropriate for browsing long lists of results.

Doulkeridis et al. present a basic approach for service discovery by matching a *user context* and a *service context*, both represented by terms [18].

The *context-sensitive service discovery system* (CSDS) as proposed by Kuck et al. [19,20] distinguishes between the set of the user's service needs and the set of semantic service descriptions. Due to the limitations in computational power of mobile devices the *user context* is represented in a simple data representation on the client (e.g. key-value). The context information in this model is categorized into *static* – mostly constant over time – and *dynamic* – changing over time – attributes. Personal information such as gender and date of birth are regarded as static attributes. Dynamic attributes can be related to the real-world situation as, for example, time and location or virtual (information) world situation as the user's documents and emails. When a service discovery request takes place the user context is passed to the discovery provider along with the query terms specified by the user.

The *service context* consists of so-called *features* that again fall into the categories *static* and *dynamic features.* The static features contain information such as purpose, provider and language that are derived from the web service description by semantic analysis of WSDL (cf. 3.4) documents which are originally meant to describe the syntactically correct usage of the Web Service's methods. The dynamic features contain relevance feedback collected from users. They include user contexts of users to whom the service was relevant, queries issued by users before invoking the service and other information that can be submitted by or extracted from the behavior of users.

Natural-language terms that are used as the instances of some of these features are limited in that their semantic relationships are usually not established in computer systems. In order to enhance the quality of the service discovery process the *service context* as implemented on a server can be represented by more sophisticated semantic models. Kuck et al. propose the use of OWL ontologies in a formal model specified in RDF.

Although the internal representation of user and service contexts is very similar to already established models (key-value and ontologies) term-based models are considered a separate context modeling approach since the matching of the two contexts is done on term level using matching algorithms that are common in information retrieval.

2.3 Context Querying and Reasoning

Querying context-information serves different purposes. The application can be either querying for its own context or the context of another well-known entity (as in CAMUS) or it can query in order to discover entities that are relevant to them because their contexts satisfy certain filter and matchmaking conditions (as in the CoOL system architecture [17] or CSDS).

In order to respond to context queries that specify higher-level contexts a reasoning or inference engine is needed. This engine should be capable of inferring higher-level contextual information by combining lower-level contextual information stored in the context repository. Logic and ontology-based data representations are preferable where context reasoning is desirable in order to process a query. In case of an ontology-based data representation the reasoning engine operates on the ontology, the context data linked to instances in the ontology and a defined set of rules. Several authors [17,3] propose the use of existing reasoning engines within the query and reasoning layer of their system architecture. Depending on the reasoning engine and the context model that backs the context-aware system a *query language* is needed in order to formulate queries.

Systems that rely on a term-based context model do not necessarily depend on semantic inference. The matching of context-enhanced service requests and service descriptions corresponds to matching the terms by which they are described. The results of this matching are ranked according to a ranking function that operates on a certain information retrieval model such as the vector space model or a probabilistic model. For a thorough discussion of information retrieval models, see [21]. As suggested by Kuck et al. in [19] , semantic modeling of the service context could be used in order to expand the service request queries.

2.4 Context Provider (Aggregation and Delivery)

Contextual information can be offered to other applications or other components within the same application by a *context provider*. The communication between the context provider and other parties is potentially bi-directional. Context information is either delivered upon incoming requests or as notification to listening parties. The context provider acts as a facade hiding the intrinsic details

of context processing from other system components. Its interface for querying contextual information as well as the data delivered at notification can thus be independent from the underlying query language, inference engine and data representation.

Composition of context provider services or an appropriate context provider discovery mechanism allow for contextual information to be distributed between several parties in the service-oriented architecture. Clients of context provider services have access to a single centralized instance in order to request contextual information. In CAMUS this instance (*context delivery service)* is not a monolithic context provider service but rather a lookup interface to discover context providers (*context aggregators)* to whom it provides registration facilities [3]. In the CoOL system architecture clients and service providers access a *context management access point* that is implemented by a potentially distributed context-management implementation [17].

3 Languages used in Context-aware Service-oriented architectures

Context-aware system architectures base their collaboration and interoperability on standardized languages and protocols. This section briefly describes some of the standards which are used by the systems we inspected. RDF(S) and OWL are part of the semantic web stack while UDDI and WSDL are used in service oriented architectures. OWL-S bridges these two worlds by providing an ontology for describing web services.

3.1 RDF and RDF/S

The *Resource Description Framework* (RDF) is one of the core elements of the *Semantic Web Stack* which was devised by Tim Berners-Lee and is advanced by the W3C [22]. Designed as a meta-data model, RDF consists of statements made about resources. These statements, called triples in the RDF terminology, occur in the form of *subject-predicate-object* expressions. The subjects and objects identify concepts and their relationship concerning a certain aspect is established by the predicate. Thus, conceptually, a set of RDF statements represents a labeled directed graph.

Unique identification of the elements of RDF is essential in order to share semantic concepts represented by RDF between agents. Therefore, subjects and predicates are resources identified by a *Uniform Resource Identifier* (URI), objects are described by another resource or a literal.

Today's standard serialization (notation) is RDF/XML [23] where resources and statements about these resources are encoded as the nodes of an XML document. Two types of nodes are defined in RDF/XML: Resource nodes represent resources and contain only property nodes, which define a predicate and contain other resource nodes or literals as objects.

Listing 1.1. An RDF example describing properties of the Wikipedia article on RDF [24]

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description
      rdf:about="http://en.wikipedia.org/wiki/Resource_Description_Framework">
    <dc:title>Resource Description Framework</dc:title>
    <dc:publisher>Wikipedia, the free encyclopedia</dc:publisher>
  </rdf:Description>
</rdf:RDF>
```

RDF is complemented by *RDF Schema* (RDF/S) [10] – a vocabulary for describing the basic semantics of resources in terms of resource classes and their valid properties. RDF by itself does not put any restrictions on the relationships it establishes between resources. The properties to be used in order to describe a resource are defined using RDF Schema. Classes of resources and generalization-hierarchies between these classes restrict the type of resources such a property can be applied to.

The above code example defines a resource `http://en.wikipedia.org/wiki/Resource_Description_Framework` that has a property "publisher". This property is defined in the RDF Schema for the Dublin Core Metadata Element Set available at `http://purl.org/dc/elements/1.1/`. The definition of such a property can restrict its applicability to a certain *domain*, for example a document in this case, and *range*, such as a person or institution. The meaning of the concepts of domain and range can be demonstrated on statements of the subject-predicate-object form, where the domain corresponds to the class of the subject, the property is the predicate and the range corresponds to the class of the object.

3.2 OWL

Another important part of the Semantic Web activity is the *Ontology Web Language* (OWL) [15], which is built on top of RDF and RDF/S on order to represent machine-interpretable semantic content. OWL is recommended by the W3C as a successor to the earlier OIL and DAML+OIL. Its core elements are *classes*, their related *properties* and *instances*. With respect to RDF/S it adds more vocabulary for the description of classes and properties and their relationships. It allows, among others, to describe the relationships of classes using Boolean expressions such as *unionOf* or *complementOf*. Restrictions can be placed on the validity of property values in class definitions. In addition to defining range and domain of properties OWL introduces the notion of cardinality of properties.

The source example in listing 1.2 is taken from W3C's sample food ontology. It demonstrates how relationships between classes are established using generalization and Boolean set operators and how restrictions can be placed to properties of these classes. According to this ontology, potable liquids are a sub-

Listing 1.2. An excerpt of an OWL-descriped food ontology [25]

```
<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="#ConsumableThing" /
  <owl:disjointWith rdf:resource="#EdibleThing" />
</owl:Class>

<owl:Class rdf:ID="Juice">
  <rdfs:subClassOf rdf:resource="#PotableLiquid" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromFruit" />
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

class of consumable things and disjoint from edible things. Juices are a potable liquid with the additional restriction to be made from at least one fruit.

According to the W3C OWL is designed explicitly for generic tool support. These tools foster reasoning about any OWL-defined ontology regardless of its subject domain. A reasoning tool is capable of inferring properties that are not directly assigned to resources in a given ontology by applying logical deductions to the classes, properties and their relationships that are defined in that ontology. Context-aware systems that are based on ontologies can use the capabilities of reasoners in order to infer higher-level context information from lower-level context information stored in a context repository. A list of available OWL tools is available at [26].

3.3 UDDI

Universal Description, Discovery and Integration (UDDI) [27] is a standard for publishing and discovering the services of a service-oriented architecture. UDDI defines XML-based registries either for use on public networks or within a company's internal service infrastructure. The standard specifies protocols for accessing the registry along with methods for controlling the access to the registry and a mechanism for distributing records to other registries.

The UDDI data model knows several data types for storing data and metadata about services. These are defined in several XML schemas. Among these, data types for describing the function of a service, information about the organization that publishes the service, its technical details including its API and various other metadata – called the tModel – are included. UDDI does, however, by itself not allow any means of semantic matchmaking between a service description and a client's service needs and does not manage any contextual information about services and the clients that use or have used these services.

3.4 WSDL

The *Web Service Description Language (WSDL)* [28] is an XML-based language providing elements for describing web services. It is defined by the W3C and is largely accepted as an industry standard. Clients can inspect WSDL files to find information about how to communicate with a web service. Web service descriptions are composed from several element types on different levels of abstraction to allow reuse of these descriptions independent of physical locations and data formats of the services.

Communication between web service clients and web services is accomplished usually by exchanging *messages* in an XML format. WSDL specifies the data types which constitute these messages, usually in the form of XML schemas. *Operations* are defined in terms of input and output messages. In order to deploy the operations in web services a *binding* to a specific communication protocol, usually SOAP [29], is declared. All of the former definitions including the binding are reusable components since they do not describe any physical communication endpoints. The actual *service* element in a WSDL consists of a collection of *ports*, physical endpoints mapped to a binding.

3.5 OWL-S

To overcome the lack of semantic description capabilities in UDDI and SOAP, OWL-S [30] was created as an OWL ontology for describing semantic web services. Its main target is to enable applications to automatically discover and invoke web services. In contrast to UDDI registries which were targeted at human service selectors, OWL-S descriptions are meant for application clients. An OWL-S ontology consists of three main parts: the *service profile* which describes the purpose or business function of the service, the *service model* which describes how clients can interact with the service by detailing the semantic inputs and outputs of the service and a *service grounding* containing more technical details on how to access a service using communication protocols.

Strang et al. consider the use of OWL-S in context-aware service oriented architectures in [17]. The authors propose the extension of its predecessor DAML-S by a *service context* containing a formal description of a service's contextual interoperability.

4 Context-awareness in Service-oriented Architectures

In this section we discuss context-aware components that have been proposed to be deployed within a service-oriented architecture. The internal architecture of proposed implementations of context-managing and -providing components in context-sensitive middleware and service-oriented architectures is presented in Sec. 4.1. Sec. 4.2 describes how these components interact with other components in a superordinate distributed or service-oriented architecture. Context-sensitive service discovery will be explored in 4.3. Its focus lies on system architectures were context is primarily used in order to improve the relevance of the results of service-discovery requests based contextual information.

4.1 Internal architecture of Context-management implementations

Context Toolkit An early approach towards facilitating the context-awareness of applications is the *Context Toolkit* published by Salber et al. [31,32]. It was motivated by the fact that sensing hardware was highly available at the time but building context-aware systems was hindered by the lack of abstraction from this hardware. The authors considered the flow of contextual data in a system as comparable to user input and hence designed their system in analogy to user interface toolkits. The basic elements in this architecture are widgets that encapsulate information about a single piece of context.

Although the Context Toolkit's architecture is rather a componentry than a layered architecture, it contains matches to most concepts introduced in section 2. Components that abstract from specific sensing hardware and acquire raw data from a single sensor or a set of closely related sensors are called *generators*.

Context information acquired by the generators is then stored in the attributes of a *widget*. This kind of context data representation corresponds to an object-oriented context data model. Widget classes encapsulate the state of a certain aspect of the system's context and define suitable operations on that state.

The Context Toolkit lacks a dedicated context provider layer. Instead, context is provided to applications or other system components by the widgets themselves, either through their programming interface or by registering callbacks. Notifying collaborators about context changes using callbacks corresponds to the communication style used in UI widgets, where the state of a widget is usually most relevant to the application when it changes. Higher-level context information that requires information from several widgets is delivered by a special type of widget called *aggregator*.

Basic context reasoning capabilities are provided to widgets and components that use widgets by *interpreters*. Since interpreters are not based on a generic semantic representation of context, implementations depend on specific widgets and the context information they provide. Dedicated querying facilities are not defined for the context toolkit. Widgets and aggregators have to be known by collaborators in order to extract their contextual information and cannot be acquired through a registry.

CAMUS A *Context-Aware Middleware for Ubiquitous Computing Systems (CAMUS)* has been proposed by a group of researchers from the Kyung Hee University of Korea in several publications [2,3,4,5,33]. The different concepts of context-management are addressed in a layered architecture encapsulated by a service that provides context delivery to other components using the middleware.

The CAMUS architecture is roughly divided into three layers: a *sensor* layer, a *knowledge processing* layer and a *delivery* layer. In the sensor layer raw data is acquired and processed by software components called *feature extraction agents* (FXA) as introduced in 2.1. In order to deploy any type of sensor within this architecture the developer defines a new FXA providing:

- The native driver in order to communicate with the sensor hardware.
- The algorithm in order to extract the most descriptive features from raw sensor data. In case of facts that are not directly observable by sensors these are composed to feature vectors that can be used as input to a pattern-recognition method.
- The meta-data describing the FXA's mapping from extracted real-world features (such as the identification number of an RFID tag) to virtual context information (such as the presence of the entity carrying the tag).

Sensor data can be requested using standardized data structures or a feature markup format. The feature-context mapping in the CAMUS architecture serves as a bridging component between the sensor layer and the knowledge-processing layer. It converts the feature tuples that are delivered by the FXAs to context information that can be managed by a context repository.

Multiple distributed domain-specific *context repositories* representing different domains such as home, office or university. A domain ontology is stored in such a context repository along with the current and historical state of the instances of the domain ontology. CAMUS uses OWL to define and store its context ontologies. Domain ontologies are organized in hierarchical structures which allow a single ontology graph from a single context repository to be integrated in a large meta-graph combining all context ontologies.

Reasoning is handled by two different types of *reasoning modules.* High-level context information that can be inferred from low-level context information by logical deductions using ontologies is provided by an ontology reasoning module. A machine learning module infers high-level context from vague or incomplete sensing data by applying machine learning techniques such as fuzzy logic, Bayesian networks or neural networks. Querying is provided by *context aggregators* which, in the CAMUS architecture, are part of the delivery layer. The initial implementation of CAMUS uses RDQL to query context data.

Context delivery, which corresponds to the concept of a context provider in our reference architecture, is divided into two separate layers in the CAMUS architecture. Context is provided by *context aggregators,* each of which performs a specific function such as querying a certain context aspect for a specific entity and delivering this context information back to an application. Context aggregators are registered through a registration interface to *context delivery services.* Applications or collaborating system components can query the context delivery services in order to obtain references to context aggregators that satisfy their contextual information needs.

CoOL The *Aspect-Scale-Context (ASC) model*, which has been described previously, is the underlying context model for the *Context Ontology Language (CoOL)* [17]. This language is proposed as part of the context management implementation of a service-oriented architecture model. The internal architecture of the context management implementation contains components which cover the main concepts of context management. The concept of sensing and a layer of abstraction for sensing are not explicitly mentioned in [17]. It is assumed,

however, that a mechanism that extracts raw data and stores it as context information is in place.

Three types of data are stored in the context repository: *facts*, *ontologies* and *rules*. Facts contain the current instance values of the concepts defined by the ontologies. Strang et al. define the use of roles "to assert inter-ontology relationships and 'complete' the ontologies by computing implicit hierarchies and relationships" [17]. The context ontologies are defined in CoOL, a language consisting of two ontology languages: OWL and F-Logic [34]. F-Logic combines knowledge representation and knowledge querying capabilities.

On the querying and reasoning layer OntoBroker [34] , an advanced reasoning engine for reasoning about ontologies defined in F-Logic is deployed. The context-provider service, titled *context-management access point* by the authors of CoOL is accessed by other components in a service-oriented architecture. Incoming requests are mapped to F-Logic queries and forwarded to the inference engine. Components that are interested in notifications about a context changes under certain conditions can register in the context-management access point. In this case a relevance condition filter is specified as an F-Logic statement.

SOCAM Another context-management architecture based on an ontology context model is the *Service-Oriented Context-Aware Middleware (SOCAM)* [35,36]. Rather than a single service façade with an underlying layered architecture SOCAM consists of *context provider services* which deliver context information on different levels of abstraction using service composition.

Raw data acquired by one or several interrelated sensing devices is encapsulated by *context providers*. SOCAM's definition of a context provider does not fully match the general high-level concept of a context provider but is on the smallest scale rather comparable to a context widget in the Context Toolkit that has an interface which is published as a service in a service-oriented middleware.

Higher-level context data is made available by *context interpreters* which act as special context provider services towards their clients and are clients themselves to context providers which encapsulate lower-level context data. Context interpreters consist of two layers: a *context knowledge-base* and a *context reasoner*. The knowledge base corresponds to the concept of a context repository. Much alike CAMUS the context is represented in terms of different domain-specific context ontologies which can be combined in a generic ontology. Ontologies are again defined in OWL which are stored in a database along with their current instance values. A context interpreter supports multiple logic reasoners such as OWL reasoners or general rule-based reasoners.

A separate *service location service* serves as a registry to the context providers and provides context-aware service discovery to client applications. Context providers specify the kinds of contexts they provide to the registry and client applications can formulate queries containing OWL expressions in order to request a service that is able to satisfy their contextual information needs. Semantic matching is applied to these OWL expressions and the ontologies advertised by the different context providers and interpreters registered with the service loca-

tion service. Therefore, the service location service corresponds to the concepts of service delivery in our reference model but also incorporates the querying facilitates which are located in a lower layer of the reference model.

4.2 Uses of Context-management within Service-oriented Architectures

Context Toolkit Widgets are the central components that deliver contextual information to applications. In order to become aware of a widget's contextual state the application can either poll it or register itself on a callback in order to be notified. To facilitate this kind of interaction within a distributed environment the Context Toolkit defines an XML-based remoting mechanism over HTTP [32]. The Context Toolkit does, however, not define any means for an application to discover new widgets in the distributed environment according to its contextual information needs.

RCSM *Reconfigurable Context-Sensitive Middleware (RCSM)* [37] for pervasive computing provides transparent ad-hoc communication between mobile application peers. It was not mentioned in the previous subsection because the information provided by the authors emphasizes on the middleware characteristics rather than the internal context management. In RCSM a *context-sensitive object request broker* (*R-ORB)* provides communication transparency by performing service discovery whenever a registered object's context expression becomes true. Objects which communicate using the middleware consist of two parts: a context-sensitive interface description and a context-independent implementation. The interface description contains the object's method signatures and context expressions. The implementation provides the operations to be triggered upon fulfillment of the context expressions. Interface descriptions and implementations are matched by deploying both into a so-called *adaptive object container* (ADC).

An R-ORB in RCSM is not a typical context provider service. [37] does not specify the internal representation and processing of the context in terms of the concepts presented in section 2. Externally its behavior differs from our concept of a context provider service since the provision of contextual information to the peers is fully transparent. An R-ORB is not yet another service within a service-oriented architecture but a central communication gateway to be passed by every communication between application clients and services.

CAMUS Context aggregators in CAMUS are the services that interact directly with application clients providing them with a defined subset of the system's overall context. In this sense it is possible to deploy them as typical services within a service-oriented architecture. Communication between application clients and context aggregators can be initiated by either side, in order to poll aggregators for contextual information , or notify application clients about context changes.

Access to context aggregators is managed by context delivery services. These services provide a registration interface that context aggregators use in order to specify the context they provide. Application clients can invoke a context delivery service's lookup interface which enables them to discover context aggregators that provide contextual information according to the client's information needs. Additionally, context delivery services provide dynamic access control mechanisms to ensure privacy and overall integrity of the system [38].

CoOL Strang et al. propose CoOL context providers as an extension to the generic service model introduced by the Munich Network Management (MNM) team [39]. In this model actors are separated in the *customer domain* and the *service domain*. Elements in this model that are shared between both domains are related to as the *abstract service* which can be fulfilled by a middleware. Strang et al. extend this model with a *context provider domain*. In this architecture, titled *MNMplusCE* by the authors, context providers are not yet another service, since they are involved as a third party service provider in the interaction between customer and service domain enabling context-aware services and context-aware service usage. Customer applications, services and middleware components get access to the context provider implementation through its *context management access point (CMAP)*. Similar other architectures presented in this paper, the CMAP supports being polled for contextual information as well as asynchronous notification of other components that specify their particular interest using F-Logic-based condition filter expressions.

SOCAM All components of the SOCAM architecture act as distributed services communicating over a common communication protocol – Java RMI in the reference implementation. *Context-aware services* is the term used for all components within this architecture which are interested in contextual information. In order to discover suitable context providers the context-aware services specify their context needs to the service locating service which in term delivers instances of potentially composed context providers. Context providers in SOCAM support again both styles of communication initiation – polling and notifying.

Comparable to RCSM ADCs context-aware services specify actions to be triggered by a set of rules whenever the current context changes. These rules, defined as first-order logic expressions, are loaded into the context reasoner of an interpreter, which will execute the actions whenever these expressions are satisfied after a change.

4.3 Context-Sensitive Service Discovery

Further advantages of enabling context-awareness in service-oriented architectures become visible in service discovery. The focus of context-sensitive service discovery methods lies on improving the effectiveness of the discovery of regular web services. Although the behavior of these web services might not be

affected by any context their relevance to clients, especially human users, might be asserted considering contextual information.

In context-sensitive service discovery the relevance of a service according to a user's service needs is determined by matching the two types of context: *service context* and *user context*. Doulkeridis et al. [18] build a context-aware service directory as an extension to standard web service registries such as UDDI. A query to the service directory consists of two parts. Q_{usr}describes a keyword-based service request containing user-specified search keywords. Q_{ctx} formulates a contextual query containing information about the user's context. The generation of Q_{ctx} is done using information from two different, regularly updated repositories: the *user profile repository* storing user preferences as part of the overall context and the *device profile repository* containing information about the device that is used to access a service. Processing these queries, the service directory will first find all services which are relevant to the user's specified query Q_{usr} in a straightforward service discovery manner. Secondly, Q_{ctx} is applied as a filter to the results induced by Q_{usr}, reducing the result set significantly to match the user's specified service needs as well as the user's context.

The general approach of context-sensitive service discovery matching user and service contexts is taken further by Kuck et al. [19,20]. Their *Context-Sensitive Service Discovery System (CSDS)* defines a formalized service discovery model and uses state-of-the-art information retrieval techniques for the matching. In classic information retrieval a query as an expression of of the user's needs is matched against a collection of documents and a ranking concerning the relevance of the documents to the user's information needs as expressed by the query is produced by a suitable ranking function. Both query and documents are often composed from language terms. CSDS transfers this model to service discovery defining a term-based context model. Since the services to be retrieved in this system are not context-aware themselves their context terms have to be extracted by the system. Static information about the service can be derived from the service description usually specified in WSDL documents by semantic decomposition of the operation and parameter names. Dynamic context information is generated from user relevance feedback. Implicit and explicit feedback from a community of users concerning the relevance of a service to the users' needs and contextual situation is fed back into the system to enrich the dynamic service context terms. An in-depth discussion of using implicit user feedback in order to improve the quality of ranking search results in a web search engine can be found in [40]. The results of the matching of user and service terms are ranked by the CSDS according to an adapted Okapi BM25 formula:

$$w(s,q,u) = \sum_{t(q \in \mathcal{PT}_{\sqcap})} w_t \cdot \frac{2tf}{dl/avdl + tf} \cdot \frac{2qtf}{1 + qtf}$$

A detailed discussion of the terms used in this formula is given in [20] and is beyond the scope of this paper.

5 Evaluation

Several proposed systems that manage context and use contextual information as an enhancement to service-oriented architectures have been introduced in the previous sections. In this section we evaluate these systems' approaches according to the following evaluation criteria:

- Architectural separation of the concepts presented in section 2 (sco) – We look at the degree of separation in terms of independent software componens or layers that implement the different concepts according to the reference model.
- Abstraction from sensor data (asd) – To what degree the accessible context representation is abstracted from the raw sensor data.
- Transparency and indirectness (tni) – We evaluate in how far the use of context providers or even the contextual information itself is transparent to the components that take advantage of the context-awareness.
- Component interaction style (cis) – Context providers and context-aware components use different styles of interaction. In the most basic cases the context-aware components need to poll the context providers for the context they are interested in. More advanced is the registration of callbacks which are triggered upon the satisfaction of filter conditions. The most advanced systems allow developers to specify in a declarative manner which operations should be invoked on a component when certain contextual conditions arrive.
- Adherence to standards (sta) – To what extent are standards such as the one discussed in section 3 being followed. This is important for being able to share contextual data representations between different systems.
- High-level context inference mechanisms (cim) – The quality and flexibility of the system in the use of inference mechanisms to derive high-level contextual information.

Table 1. Evaluation results for the inspected systems.

	sco	*asd*	*tni*	*cis*	*sta*	*cim*
Toolkit	-	+	-	-	-	-
RCSM	-	-	++	++	-	-
CAMUS	++	++	-	-	+	++
CoOL	+	++	-	+	+	+
SOCAM	+	+	+	++	+	-/+
Doulkeridis					+	-
CSDS					+	+

The systems we inspected had different goals and thus do not put the same amount of emphasis on fulfilling all criteria and concepts. It is therefore not possible to argue that a single approach is superior to the others in that it is

more appropriately applying context-awareness to service-oriented architecture. Toolkit is a simple approach that certainly can be seen as an ancestor of many later systems since it is often referenced by these. The more advanced systems in terms of providing context-management to service-oriented architectures are CAMUS, CoOL and SOCAM. RCSM provides the highest degree of context transparency but lacks an elaborate model of context management. It is admittedly hard to apply these criteria to the service-discovery systems presented in this paper. It can be argued though that CSDS is much more promising than the system presented by Doulkeridis because it has a more elaborate model of user and service contexts and also because the matching is done using state-of-the art methods from information retrieval. The middleware-oriented context-management systems we inspected in sections 4.1 and 4.2 will be evaluated in more detail in the following sections. Table 1 shows an overview of our evaluation results.

Context Toolkit as an early approach does not explicitly mention service-orientation but the way it allows widgets to be used alone and in a composition and the XML-based remoting capabilities can be regarded as a basic form of service-orientation already. It does, however, not clearly separate the concepts of context management since it lacks a standard mechanism for reasoning and context delivery. It is possible to abstract from sensor data using the separation of generators and widgets but there is no architectural concept that transforms raw data into contextual information in a generic way. The degree of transparency of clients and context providers is very low because the clients have to explicitly deal with the context. In addition, there is no facility for discovering context providing components such that clients have to have explicitly knowledge of context providers they use. The component interaction styles are basic – polling and callbacks are supported but there are no mechanisms specifying semantic relevance conditions besides listening to named callbacks which are particular to singe widgets. Besides using an HTTP and XML based remoting protocol the Context Toolkit does not use any of the above standards. High-level contexts may be derived by composing several widgets but this functionality would have to be implemented by the aggregators individually. Semantic inference of higher-level contexts and would be hard to accomplish with this low degree of formality in the context representation.

The main focus of RCSM lies on providing an ORB-based middleware infrastructure where context is fully transparent to the distributed objects (which could in a broader sense be considered services). The context-management is not described in detail. It thus lacks a clear separation of context-management concepts. There might be abstraction from sensor data in RCSM but this is not explained by its authors. It is, however, certainly the most advanced system in terms of indirectness and transparency, since the implementations of the distributed objects never explicitly request or handle contextual information. Therefore, interactions take place only between the middleware and the object containers upon the satisfaction of certain context conditions. Standard lan-

guages are not used by RCSM. Sophisticated semantic context reasoning is also not described for RCSM.

CAMUS takes the most thorough approach to separating the concepts of context management providing detailed descriptions on all of the concepts and separating them into a layered architecture. The abstraction from sensor data is very high and increases from lower to higher levels. Transparency in the use of contextual information is low, since clients using contextual information have to explicitly declare which contexts they are interested in and therefore have to explicitly process notifications from the context aggregators. The component interaction style supports polling as well as registering notifications on the aggregators. It is however not discussed by the authors how relevance filters can be applied to the notification mechanisms of context aggregators. CAMUS uses standard languages such as OWL for describing and RDQL for querying ontologies. The reasoning facilities in CAMUS are highly extensible. In addition to allowing various ontology reasoners to be plugged into the architecture it is also possible to use machine learning techniques in order to derive contextual information from vague or incomplete input data.

CoOL is mainly described as a joint language for representing contextual information. Its authors also propose a middleware approach for bringing context-awareness to service-oriented architectures. The concepts of context management are well separated. The authors, however do not mention how the context data repository interacts with the sensor data. The abstraction from raw sensor data is very high due to the formal ASC models which allows for conversion between data on different scales. Clients using contextual information provided by CoOL have to explicitly deal with the contextual information leading to a low rating for this criteria. The interaction style is advanced. Clients can specify the contextual information that is relevant to them using the F-Logic language leading to relevant notifications only. But clients still have to decide actively in what way their behavior is influenced by the context change. CoOL also makes use of the OWL standard. For some tasks it uses F-Logic but claims to be able to interconvert these two languages. For high-level context inference it relies on a single but very powerful inference engine.

SOCAM is the system which most explicitly emphasizes the use of a service-oriented architecture. For SOCAM, the concepts of context-management can be separated into different services where each higher-level service is a client to a lower-level service. This separation is, however, not enforced by the architecture since arbitrary clients could in principle access lower-level context-aware services. Abstraction from sensor data is mainly provided by the so-called interpreters. These integrate multiple lower-level context-aware services in order to reason about context. The indirectness of the communication between clients and context providers is low, since they clients interact directly with located context providers. Contextual information can still be used in a transparent matter because SOCAM supports a special style of interaction. In addition to polling and notifying, SOCAM supports the registration of first-order-logic statements on the interpreters which cause an operation on a service to be called upon the ar-

rival of a certain context condition. As a consequence the implementation of this service can be completely unaware of the context management itself. SOCAM is another system that uses OWL for context representation but complements it with its own OWL-like query language and a first-order-logic language, which uses OWL resources to denote contextual facts. SOCAM provides high-level interference mechanisms based on ontologies but also allows their circumvention by giving direct access to context providers.

6 Conclusion

In this paper we presented and evaluated several systems that can be applied to combine context-awareness and service-oriented architectures. From these systems we derived common concepts of context-management which are driven by different types of context models. These concepts were proposed as layers for a layered reference architecture. The inspected systems were then evaluated according to their fulfillment of the reference model and other criteria we defined for asserting the appropriateness of systems for context-awareness and the service-oriented paradigm.

The list of systems we presented is not meant to be a complete reference to context-awareness and service-oriented architectures. We chose some exemplary and well-documented systems in order to demonstrate possible approaches in-depth on a limited number of systems. Future systems might build upon the foundations laid by the ones presented in this paper to use context-awareness in service-oriented architectures in a flexible and transparent way. Context-management in large service-oriented environments, however, remains a challenging task. The systems presented in this paper provide technical frameworks – the challenge lies in applying them to intra- or even interorganisational environments. The difficulties of providing comprehensive ressource descriptions and ontologies are apparent in the relatively slow proceedings of the Semantic Web Initiative. The rather diffuse Web2.0 movement proposes more pragmatic approaches for meta-data generation relying on the collaborative power of its communities. This is comparable to the term-based context models used by CSDS. These methods however, aim at narrowing a set of choices to be made by a human user and are not suitable for decisions to be made by computers in a service-oriented environment. Future research has to provide methods for collaborative creation and management of formal context models that form the core of high-level context-aware systems.

References

1. Schmidt, A., Beigl, M., Gellersen, H.W.: There is more to context than location: Environment sensing technologies for adaptive mobile user interfaces. In: Proceedings of Workshop on Interactive Applications of Mobile Computing (IMC'98). (Nov 1998)

2. Ngo, H.Q., Shehzad, A., Kiani, S.L., Riaz, M., Lee, S.: Developing context-aware ubiquitous computing systems with a unified middleware framework. In Yang, L.T., Guo, M., Gao, G.R., Jha, N.K., eds.: EUC. Volume 3207 of Lecture Notes in Computer Science., Springer (2004) 672–681
3. Ngo, H.Q., Shehzad, A., Ngoc, K.A.P., Lee, S.Y., Jeon, M.: Research issues in the development of context-aware middleware architectures. [41] 459–462
4. Shehzad, A., Ngo, H.Q., Lee, S.Y., Lee, Y.K.: A comprehensive middleware architecture for context-aware ubiquitous computing systems. In: ICIS '05: Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05), Washington, DC, USA, IEEE Computer Society (2005) 251–256
5. Kiani, S.L., Riaz, M., Zhung, Y., Lee, S., Lee, Y.K.: A distributed middleware solution for context awareness in ubiquitous systems. [41] 451–454
6. Strang, T., Linnhoff-Popien, C.: A context modeling survey. In: Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham, England (September 2004)
7. Lee, C., Helal, S.: Context attributes: An approach to enable context-awareness for service discovery (2003)
8. Kiss, C.: Composite capability/preference profiles (CC/PP): Structure and vocabularies 2.0. W3C working draft, W3C (April 2007) `http://www.w3.org/TR/2007/WD-CCPP-struct-vocab2-20070430`.
9. Wapforum: User Agent Profile (UAprof). `http://www.wapforum.org/what/technical/SPEC-UAProf-19991110.pdf`
10. Tobin, R.: An RDF schema for the XML information set. W3C note, W3C (April 2001) `http://www.w3.org/TR/2001/NOTE-xml-infoset-rdfs-20010406`.
11. Ayed, D., Berbers, Y.: UML profile for the design of a platform-independent context-aware applications. In: MODDM '06: Proceedings of the 1st workshop on MOdel Driven Development for Middleware (MODDM '06), New York, NY, USA, ACM Press (2006) 1–5
12. Halpin, T.: Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design (The Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann (2001)
13. Henricksen, K., Indulska, J., Rakotonirainy, A.: Generating context management infrastructure from context models. In: 4th International Conference on Mobile Data Management (MDM), Industrial Track Proceedings, Melbourne (January 2003) 1–6
14. Uschold, M., Grüninger, M.: Ontologies: principles, methods, and applications. Knowledge Engineering Review **11**(2) (1996) 93–155
15. van Harmelen, F., McGuinness, D.L.: OWL Web Ontology Language Overview. W3C recommendation, W3C (February 2004) `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.
16. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American (May 2001)
17. Strang, T., Linnhoff-Popien, C., Frank, K.: CoOL: A Context Ontology Language to enable Contextual Interoperability. In Stefani, J.B., Dameure, I., Hagimont, D., eds.: LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003). Volume 2893 of Lecture Notes in Computer Science (LNCS)., Paris/France, Springer Verlag (November 2003) 236–247
18. C. Doulkeridis, N.L., Vazirgiannis., M.: A system architecture for context-aware service discovery. Electr. Notes Theor. Comput. Sci. **1**(146) (2006) 101–116

19. Kuck, J., Reichartz, F.: A collaborative and feature-based approach to context-sensitive service discovery. In: 16th International World Wide Web Conference, Workshop on Emerging Applications for Wireless and Mobile Access (MobEA V), Banff, Alberta, Canada (May 2007)
20. Kuck, J., Gnasa, M.: Context-sensitive service discovery meets information retrieval. In: Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on, White Plains, NY, USA (March 2007)
21. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley (May 1999)
22. W3C: World wide web consortium. Website `http://www.w3.org`.
23. Beckett, D.: RDF/XML Syntax Specification (Revised). W3C recommendation, W3C (February 2004) `http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/`.
24. Wikipedia: Resource description framework — Wikipedia, the free encyclopedia. Website (2007) Available online at `http://en.wikipedia.org/wiki/Resource_Description_Framework`; visited on May 24th, 2007.
25. W3C: Food ontology. Web resource Available online at `http://www.w3.org/TR/owl-guide/food.rdf`; visited on May 24th 2007.
26. W3C: OWL implementations. Website Available online at `http://www.w3.org/2001/sw/WebOnt/impls`; visited on May 27th 2007.
27. OASIS: UDDI Specifications TC. Website Available online at `http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm`; visited on May 27th 2007.
28. Christensen, E., Curbera, F., Meredith, G., Weerawana, S.: Web services description language (WSDL) 1.1. W3C ackowledged submission, W3C (March 2001) `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.
29. Mitra, N., Lafon, Y.: SOAP version 1.2 part 0: Primer (second edition). W3C recommendation, W3C (April 2007) http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.
30. et al., D.M.: OWL-S: Semantic Markup for Web Services. W3C member submission, W3C (November 2004) http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.
31. Salber, D., Dey, A.K., Abowd, G.D.: The Context Toolkit: Aiding the Development of Context-Enabled Applications. In: CHI. (1999) 434–441
32. Dey, A.K., Abowd, G.D.: The Context Toolkit: Aiding the Development of Context-Aware Applications. In: Workshop on Software Engineering for Wearable and Pervasive Computing (SEWPC). (2000)
33. Moon, A., Kim, H., Kim, H., Lee, S.: Context-aware active services in ubiquitous computing environments. ETRI Journal **29**(2) (April 2007) 169–178
34. Decker, S., Erdmann, M., Fensel, D., Studer, R.: Ontobroker: Ontology-based access to distributed and semi-structured unformation. In: Database Semantics: Semantic Issues in Multimedia Systems. (1999) 351–369
35. Gu, T., Pung, H.K., Zhang, D.Q.: A service-oriented middleware for building context-aware services. J. Netw. Comput. Appl. **28**(1) (2005) 1–18
36. A middleware for building context-aware mobile services. Volume 5. (2004)
37. Yau, S.S., Karim, F., Wang, Y., Wang, B., Gupta, S.K.S.: Reconfigurable context-sensitive middleware for pervasive computing. Pervasive Computing, IEEE **1**(3) (2002) 33–40

38. Riaz, M., Kiani, S.L., Lee, S., Han, S.M., Lee, Y.K.: Service delivery in context aware environments: Lookup and access control issues. In: RTCSA '05: Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), Washington, DC, USA, IEEE Computer Society (2005) 455–458
39. 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**(4) (December 2001) 297–306
40. Agichtein, E., Brill, E., Dumais, S.: Improving web search ranking by incorporating user behavior information. In: SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, New York, NY, USA, ACM Press (2006) 19–26
41. 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2005), 17-19 August 2005, Hong Kong, China. In: RTCSA, IEEE Computer Society (2005)