Strategies for On-The-Fly Composition of Context Information Services

Michael Krause, Iris Hochstatter
Munich Network Management (MNM) Team
Department of Informatics
Ludwig-Maximilians-University Munich
Oettingenstr. 67
80538 Munich, Germany
{Michael.Krause|Iris.Hochstatter}@ifi.lmu.de

Abstract

Context-aware services adapt to ever-changing environments and therefore face the challenge to acquire context information in a priori unknown situations. They rely on other services (e.g. for context acquisition) and failures of these services may occur. In this paper we introduce concepts for the CoCo Infrastructure in order to deal with services that fail to deliver requested context information. The abbreviation *CoCo* stands for "Context Composition" and is a service-oriented infrastructure supporting context-aware services that is currently developed by the MNM-Team. This composition of context information services often has to be carried out dynamically at run-time as it depends on the user's context which services can be used. We show how static instructions and dynamic methods can be combined to find a tradeoff between the level of sophistication and necessary resources, such as e.g. computing time.

Keywords

service composition, context-awareness, context provisioning infrastructure

1. Introduction

In a world with a rapidly increasing number of networked computing devices, increasing complexity of software and available digital information there are two important requirements arising. First the imperative to make sensible use of these new resource that allow highly customized services. Second the necessity to enable the user to deal with the steadily increasing complexity of services around him. Both requirements imply the need for a wide automation of context-aware services.

Context-aware Services support the user, because they adapt their behaviour and results according to automatically acquired information about the user's context. Context information can be any information that characterises the situation of the user, for example the user's location, her current activity and the technical capabilities of the mobile device used [9]. The user herself does not have to give a lot of input explicitly anymore. She does not even need to know the information. Furthermore, the service can use this information to derive additional, more significant information automatically. For example, the user can be provided with a reminder to head for the airport when information

from her calendar is compared with the flight schedule, its current distance to the airport and information about the local traffic. This saves time and reduces the complexity of the interaction between user and service. It also improves the service result as it becomes highly customized.

Because nearly all information imaginable can become context information for a specific task there is a vast amount of potential context sources that naturally are very heterogeneous. There is a wide variety of context types (e.g. location, activity, etc) kinds of sources (physical sensors, databases, etc.) and besides that the cooperation between different actors (service providers, network operators, etc.) is assumed. It usually cannot be specified during development of a context-aware service to which context information services it shall be bound, because this decision is contextual itself. For example the decision about which service can deliver weather data depends on the users current location. This is one reason why context-aware computing especially has to cope with the problem of interoperability between *context-aware services (CASs)* and services that offer context information which we call *context information services (CISs)*. The incorporation of context in the service provisioning process requires supporting infrastructure. Mechanisms for sensing, refinement and administration of context information have to be in place so that programmers and service providers can use them in a transparent way.

Compared to other concepts this one especially has to deal with the high risk of failures within the process of context provisioning. Because of the highly dynamic environment in which a CAS might be forced to use different CISs everytime new context information is needed. There might be no appropriate service discoverable, a discovered service might not answer or deliver only incorrect information. The risk of failure is eminent because the service binding can only happen at run-time and the respective services are unknown beforehand. In this paper we propose strategies to improve the chance of success for context provisioning infrastructure services. For this purpose we rely on semantic knowledge about the context in order to find workarounds.

The paper is structured as follows: In the next section, we discuss existing technologies for service composition. In section 3, the CoCo Infrastructure and CoCoGraphs as the underlying concepts for context composition in our approach are explained. Based on this, we present our work done on strategies for dynamically composing CISs in section 4 and discuss them. We conclude in section 5 with our main findings and give an outlook on future work.

2. Related Work

Composing Context Information Services can be looked at as service composition in general. Further challenges arise through the necessity of adhering to sufficient quality of context in the end and the difficulties in expressing semantics. With the term *quality of context (QoC)* we refer to quality information of context such as e.g. precision, probability of correctness, trust-worthiness, resolution, and up-to-dateness (cp. [3]). QoC can be critical for the provision of a service as the context information influences the output and/or the execution of the service. For example, outdated weather information can guide a user to a beergarden though it started raining. As context information can basically be

any information it is hard to find a common understanding about it, which is necessary for service interoperybility and service composition [14]. We will first look into existing technologies for service composition and evaluate whether they meet our requirements.

As the major objective of Web Services technology is interoperability between services developed in different programming languages on heterogenous platforms, service composition is a key feature and interfaces to the services are highly standardized [11]. The *Simple Object Access Protocol (SOAP)* is used to transmit standardized XML documents and thus enables the exchange of information between different platforms. The *Web Service Description Language (WSDL)* describes the interface of Web Services. Input and output parameters, the structure of the function, the signature, and protocols are specified in WSDL. *Universal Description, Discovery, and Integration (UDDI)*, the third technology used for Web Services, offers world-wide directories for Web Services where WSDL documents of existing Web Services are stored and can be retrieved for service composition. In [15] van der Aalst evaluates some of the different standards for Web Service composition such as BPEL4WS, XLANG, WSFL, XPDL, Staffware, MQ Series Workflow, Panagon eProcess, and FLOWer. Some of those approaches have been combined or vanished in the meantime. Currently, the most important language for Web Services composition is the *Business Process Execution Language for Web Services (BPEL4WS)*. It is very well able to model business processes and thereby combine different Web Services. Unfortunately, BPEL is not designed for dynamic changes, BPEL documents cannot be changed during run-time, and it is thus unsuitable for our purposes. Other Web Services orchestration languages (cp. [12]) like the *Business Process Modeling Language (BPML)*, the *Web Service Choreography Interface (WSCI)* and the *ebXML Business Process Specification Schema (BPSS)* are comparable in this point, they do not support dynamic reorganisation likewise.

Independent of Web Services, there exists a variety of other approaches for service composition which are not considered further here, for more information see [10], [13] and [4]. In the following, we will briefly look at Ninja, Solar, and iQL, three approaches focused on the composition of pervasive data respectively context information. Service composition in *Ninja* is done with Ninja Paths (cp. [6]). Services have to register themselves with their capabilities in terms of structural and semantic information about their inputs and outputs with a *Service Directory System (SDS)*. From the description, it can be deduced whether services can be combined or not. The Ninja path is an ordered sequence of services that results in the desired complex service, it is created using a shortest path search over the registered services. Chadrasekan et al. state that it is not possible to chain services without semantic knowledge and in their approach rely on a simple *type system* based on a XML grammar. The semantic information used in Ninja is not sufficient to express the complex semantics for CASs. Solar is an infrastructure for context-aware applications that delivers context information [7]. It provides a composition language, allowing applications to construct a graph of operators to compute desired context from appropriate sources. Graphs model the collection, aggregation and dissemination of context information and information sources produce events to which applications may subscribe. However, Solar distributes the context processing within one administrative domain and thus restricts its use heavily. A declarative language named *iQL* was developed by Cohen

et al. ([8]) to assist context-aware applications in the acquisition and processing of context information. iQL expresses requirements on the data source rather than the source itself and a runtime system discovers and binds the sources. This concept facilitates the dynamic binding of sources at run-time. Nonetheless, the language uses a proprietary, non-XML syntax and it can only bind context types that were specified during the design phase as it does not use a strong typed system like Ninja nor ontologies (cp. [6] and [14]).

3. The CoCo Infrastructure

The CoCo Infrastructure, depicted in figure 1, acts as middleware between services requesting and services offering context information. It therefore handles tasks like discovery, accounting, security or bundling and dissemination of context information. Amongst others this infrastructure offers the service of providing context information. This service is a compound service, made up of several basic infrastructure services working together. In charge of the flow control of this compound service is the CoCoGraph Controller that also features the according service access point. A CAS can access the context information provisioning service by sending a CoCo document that contains its requests for context information in form of the XML representation of a CoCoGraph.

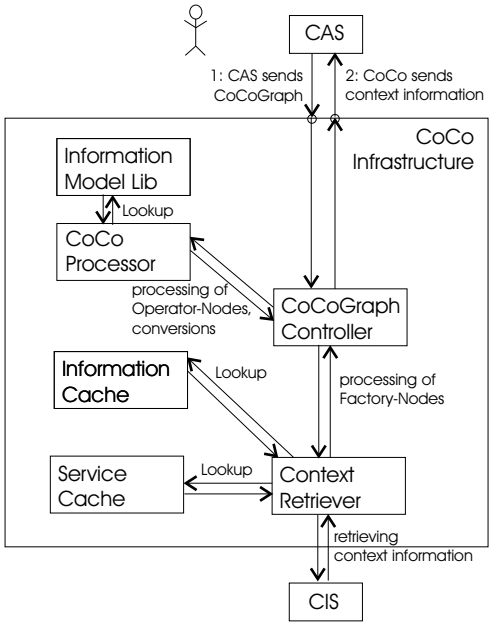

Figure 1: CoCo infrastructure

The concept and syntax of CoCoGraphs is described in greater detail in [2], we only describe the most important ideas here. Basically a CoCoGraph is made out of two main types of nodes: FactoryNodes that describe the wanted context information and OperatorNodes that describe how the context information items are to be processed to achieve high-level context information (see figure 2). CoCoGraphs are processed in the CoCo Infrastructure, see figure 1. The CoCoGraph Controller parses these nodes. Whenever it parses a FactoryNode, it requests this single information item from the Context Retriever, and whenever the Controller parses a OperatorNode it requests the CoCo Processor to execute the described operation. Furthermore the Controller observes the costs of the overall information retrieval, the quality of each information item and by doing so enforces the guidelines given in the CoCoGraph.

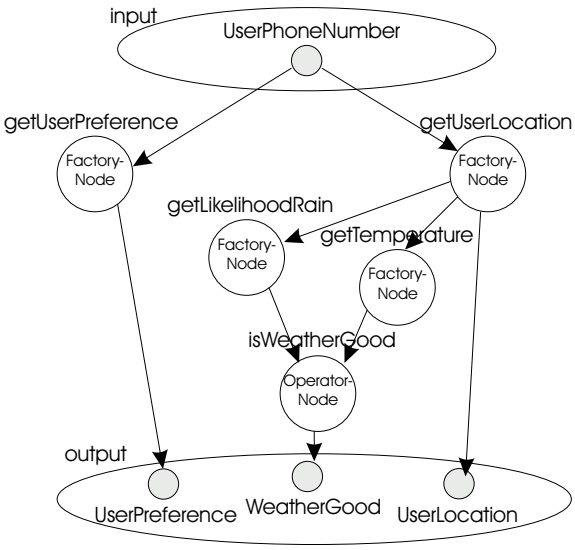

Figure 2: Example CoCoGraph of a restaurant finder service.

Figure 2 shows the CoCoGraph for an example service. A restaurant finder services needs information about user-specific restaurant preferenes, his location and the weather there. For each request for context information, the ContextRetriever seeks for appropriate services offering context information. It selects one, accesses this service and returns the received information to the CoCoGraph Controller. Both the description of all discovered services and the information itself are cached for later reuse.

The CoCo Processor executes explicit operations that are needed to derive a specific piece of context information from one or several other pieces of context information. For example it can use the information about the position of two entities to compute the distance between them. For this and similar operations the CoCo Processor relies on semantic knowledge about the information. Amongst other, this allows for executing basic operations like ordering, selection or aggregation. We call operations prescribed in OperatorNodes explicit. On the other hand there are conversion operations that are necessary everytime when a piece of information is available but in the wrong format. For example if you want to get weather information about a location that is described in longitude and latitude, but only weather information services are available that deliver information for locations that are described in address-format. The necessary conversion is also done by the CoCo processor on behalf of the CoCoGraph Controller and is called implicit, because there is no according OperatorNode present.

As mentioned above, the CoCo Infrastructure relies on the existence of a semantic-aware information model. The discovery of CISs is done by comparing service descriptions that do not only contain name of service and signatures of available methods, but also semantics of the service functionality. Only by this the discovery and composition of services can be done automatically. This knowledge about the semantics of information and service functionality is also the key for the ability to deal with failures automatically (cp. outlook in section 5).

So far we presented the basic ideas and the functionality of CoCo, the components of the infrastructure and their interactions. As discussed in the introduction, in dynamic environments it can happen that some context information services are not available and CoCo has to find equivalent ones. Therefore we need strategies for the on-the-fly composition of CISs. We follow Chakraborty and Joshi [5] who identified six key issues that are important

for dynamic service composition. First and most important of all, *service discovery* has to take place, CISs offering a particular piece of context information have to be found. In our approach CoCo, the ContextRetriever is responsible for searching appropriate CISs. Coordination of the different involved services is named *service coordination and management*, it becomes difficult when the the entities are distributed across the network. As we assume multi-provider environments where multiple organisations work together in providing a CAS, we have a highly distributed environment. Service coordination and management in CoCo is carried out by the CoCoGraph Controller. Service composition also relies on a *uniform information exchange infrastructure* meaning that services have to interoperate independently of their information exchange mechanism (e.g. remote method invocation, message-passing and the like). In CoCo, the ContextRetriever has the ability to access CISs with different methods, e.g. Web Services. As outlined in the Introduction, *fault tolerance and scalability* are crucial in context provisioning as it is very likely that CISs are not available all the time and they depend on the user's context themselves. In this paper, we present our strategies for fault tolerance for dynamic CIS discovery in section 4. Scalability is addressed in CoCo through proactive caching of both the description of discovered services and the information itself. Thus, at a later point in time, the information can potentially be used again or at least the service description is available. Also, the use of CoCoGraphs and parts of CoCoGraphs enhances the system as similar services can reuse the descriptions. Last but not least, *adaptiveness* is a key success factor to service composition. It is closely related to fault tolerance as adaptiveness here refers to the changing availability of services. Service come up and go down frequently, especially in a mobile environment. In CoCo, the discovery of context information is related to the context of the user, the system is thus inherently adapting to the environment.

4. Adapting the Graph

Often a certain piece of context information cannot be retrieved directly and has to be deduced from other context information. For example: If there is no appropriate sensor that is able to sense the distance between two specific objects, then this information must be deduced from the information about the two locations. Instead of only providing single pieces of context to the CAS, the idea of CoCo is to relieve the CAS from the whole process of context refinement including the deduction or composition of context information. Therefore the CAS delivers a CoCoGraph that describes the necessary composition to the CoCo infrastructure.

This is the key for further benefits: With semantic knowledge about the necessary piece of context information and its construction plan, the CoCo infrastructure is able to fulfil the whole process even if single steps fail. The idea is to replace bad nodes with other nodes (or with complete graphs that are inserted into the original graph where the bad node has been) without changing the result of the process. Or, if this is not entirely possible, with the best available approximation of the aimed result. Although it is possible that the processing of operations on context information fails (OperatorNodes), we assume that the bad nodes are FactoryNodes and therefore represent requests each for a specific piece of context information.

Here it is important to state the main difference between a CoCoGraph and other service composition instructions, for example those that could be expressed with the Business Process Execution Language BPEL (cp. [1]). The CoCoGraph does not describe the composition of specific *services* itself (means: the dependencies and the parallel and sequential execution of specific services) but the dependencies of specific context information items. With the CoCoGraph, the appropriate service is not selected until run-time, and perhaps for a specific information request there might be no service available at all. Compound services that involve other services need to adapt composition and selection of services when there are significant changes in the environment, whereas the mapping of services onto the composition instruction of a CoCoGraph is adapted with every run. This mapping is not possible even right before the processing of a CoCoGraph, because the selection which service is able to provide a specific information cannot be done before the preceding information items are available. For example: You can not select a weather service before you know the location it shall provide weather information for.

This is why the adaptation of a CoCoGraph in case one or several requests for context information could not be fulfilled is time-critical and must be done at run-time. There are two main tasks to be carried out:

Task 1: Decide which part of the graph is to be replaced

To achieve results that are as much alike the intended results of the unadapted graph it is an obvious policy to leave as much of the graph unchanged. Therefore at first we try to replace just the single node for which no working service could be found (see figure 3). Only if these replacement attempts fail, we increase the scope to the next-higher level: the smallest subset of the graph that has been explicitly defined (could be done with the InnergraphNode, see [2]) that contains the bad node. If the replacement attempts of one after another of these subsets fail, (or if no subsets are defined at all), the last replacement attempt is done for the whole graph.

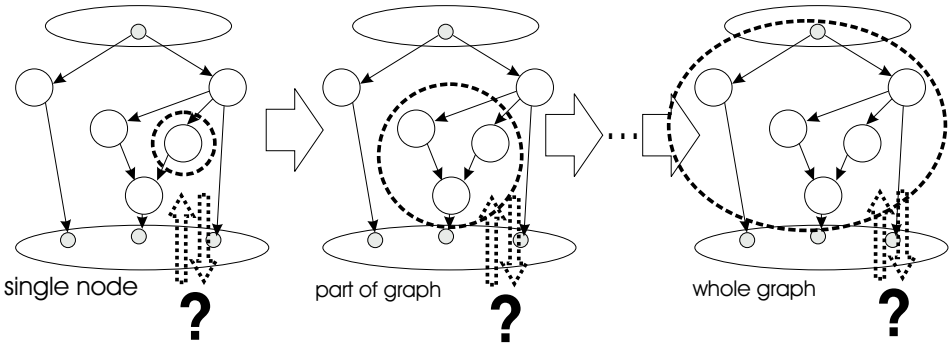

Figure 3: Selection of replacement target.

Task 2: Decide which nodes are to be inserted into the now empty place

There are three possibilities for the CoCoGraph Controller to receive the knowledge with which nodes the bad nodes could be equally or approximately replaced. Due to the

requirements for this procedure - time-critical, best possible results for the client service - we choose those knowledge base first that promises to meet these requirements best.

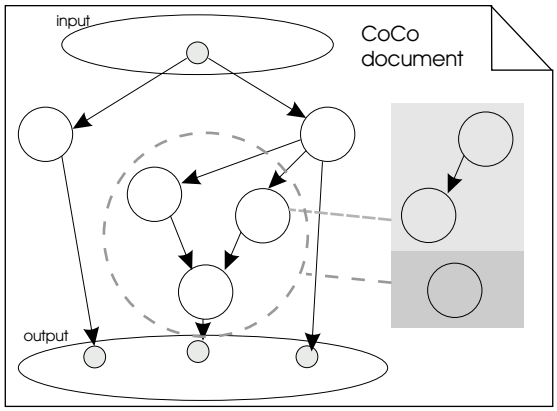

Figure 4: CoCoGraph and replacement instructions inside the CoCo document.

The developers of the CAS who designed the service and the CoCoGraph could have stated the fallback-solutions for single nodes or parts of the graph themselves. Then these solutions have to be expressed **inside the CoCoGraph** (see figure 4). If such solutions are available they are the first choice. On the one hand no additional computing is necessary and on the other hand CAS-developers know best with which results their CAS can deal best. For example, if no current weather data about the user's location could be retrieved, then the restaurant finder could assume that the weather is good enough for beer gardens if it is June, July or August, and assume bad weather otherwise.

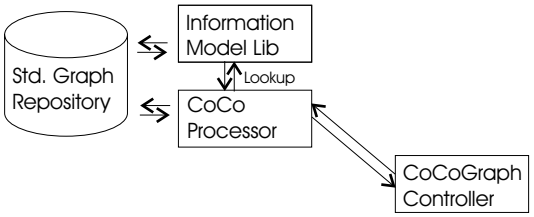

Figure 5: Replacement instructions as part of the standard graph repository.

Besides this, the CoCoGraph controller can use a **repository of standard CoCoGraphs** each of it with references to the context information classes (CI-classes, for example "temperature") it can replace or approximate. This repository is an enhancement of the Info Model Lib and can be accessed via the CoCo processor (see figure 5). For example an information about a certain location (like weather information) can be approximated with the equivalent information about locations nearby. The repository of standard graphs is also useful for CAS developers that can often simply reference such a standard graph instead of creating own CoCoGraphs, which also reduces network traffic.

Finally (if the solutions stated above did not work or are not available) the CoCo controller can use the semantic knowledge of the CoCo Infrastructure stated in the **Info Model Lib** to compute replacement nodes or graphs for the current problem on-the-fly. Like this, solutions can be found even for unusual problems. But it is obvious that this reasoning is the most time-consuming strategy and therefore only could be ultima ratio. How promising this strategy could be at all is depending on power and definition of the appropriate information model - this is out of scope of this paper (see section 5). It is possible that this reasoning is joined with repeating scans of the context retriever to learn

which context information on which entities is available at all in order to implement a kind of breadth-first-search-algorithm to find a semantically correct and as short as possible way around the bad node. This would improve the result of this strategy in time, traffic and discovery costs.

Each of these strategies, ”CoCoGraph fallback”, ”standard-graph” and ”reasoning”, can possibly provide several solutions that might fail as well as the original, unadapted CoCoGraph. After a failure, the next solution of the current strategy is tried. If they all fail, then recursively this adaptation algorithm is done for the first one (if it consists of more than one node), if it has no success, it is done for the next solution and so on. If no solution of the current strategy succeeds then the controller switches to the next one (in the order stated above), until all strategies have failed, the allowed time has run out (could be defined inside the CoCoGraph), or the adaptation algorithm has found a successful way around the bad node.

As presented, we propose different strategies for the on-the-fly composition of CISs which can be combined to obtain optimal results regarding performance and fault tolerance. In case a necessary piece of information is not available firsthand from a context information service, we use this algorithm to replace the missing parts by similar ones. There are different criteria to classify those algorithms such as e.g. directory-based vs. dynamic service discovery, expressiveness vs. performance, quality vs. costs, and others. As context retrieval is very time critical, fast work-arounds are to be preferred over time-consuming, perhaps better fitting ones. Therefore we use a substitution in the CoCoGraph itself whenever possible. This ensures the correct semantics of the replacement and is the fastest way around bad nodes. The ”standard-graph” strategy is a good trade-off between performance and expressiveness as it is available for the CoCo controller from the Info Model Lib directly. Problems with different understanding about the output can be solved using an appropriate context information model. This standard library of CoCoGraphs will be very helpful to CAS developers and could even be exchanged between different operators. Last, fitting CISs could be found out using reasoning on the available information. This strategy is prone to errors and time consuming, but it is highly flexible as it does not need any information at first hand.

5. Conclusion and Future Work

In this paper, we introduced a concept for adapting composition instruction (CoCoGraphs) for context information in case a necessary piece of information is not available firsthand from a context information service. Therefore we proposed a multi-level algorithm to replace the missing parts by similar or equivalent ones. This algorithm builds upon a semantic-aware information model as it includes dynamic reasoning methods besides static solutions. So this concept enhances the automation capabilities of the context retrieving process for context-aware services. As a result it strengthens CASs in their effort to support the user in her tasks in a proactive way.

Up to now we already implemented the basic functionalities of the CoCoGraph Controller and deployed a restaurant finder service onto the CoCo infrastructure as proof of concept. We are currently developing a meta model for context information that can

specify context information with respect to their semantics. In parallel we are going to implement the adaptation algorithm proposed in this paper with regard to this semantic knowledge. As the meta model becomes more apparent we will gradually develop the components of the infrastructure and their interaction, like the service discovering context retriever that shall work with service descriptions that include semantic information. The next step then will be to combine the CoCo concept with policies to enhance flexibility and automation in the whole system.

6. Acknowledgements

The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of the paper. The MNM Team directed by Prof. Dr. Heinz- Gerd Hegering and Prof. Dr. Claudia Linnhoff-Popien is a group of researchers at the University of Munich, the Munich University of Technology, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. For further information see http://wwwmnmteam.ifi.lmu.de.

References

[1] Tony Andrews, Francisco Curbera, Hitesh Dholakia, and Yaron Goland. Specification: Business process execution language for web services version 1.1, 2003.

[2] Thomas Buchholz, Michael Krause, Claudia Linnhoff-Popien, and Michael Schiffers. Coco: Dynamic composition of context information. In *Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous)*, 2004.

[3] Thomas Buchholz, Axel Küpper, and Michael Schiffers. Quality of Context: What It Is and Why We Need It. In *10th Workshop of the HP OpenView University Association (HPOVUA'03)*, Geneva, Switzerland, July 2003.

[4] Dipanjan Chakraborty and Anupam Joshi. Dynamic service composition: State-of-the-art and research directions. Technical Report TR-CS-01-19, University of Maryland, Baltimore, December 2001.

[5] Dipanjan Chakraborty and Anupam Joshi. Dynamic service composition: State-of-the-art and research directions, technical report tr-cs-01-19. Technical report, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD, 2001.

[6] Sirish Chandrasekaran, Samuel Madden, and Mihut Ionescu. Ninja paths: An architecture for composing services over wide area networks. CS262 class project writeup, Computer Science Division, University of California, Berkeley, 2000.

[7] Guanling Chen and David Kotz. Context aggregation and dissemination in ubiquitous computing systems. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pages 105–114. IEEE Computer Society Press, June 2002.

[8] N. Cohen, P. Castro H. Lei, J. Davis II, and A. Purakayastha. Composing pervasive data using iql. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applcations (WMCSA 2002)*, Callicoon, New York, June 2002.

[9] A.K. Dey. *Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.

[10] Gabrijela Dreo Rodošek. *A Framework for IT Service Management*. Habilitationsschrift, Ludwig-Maximilians-University, Munich, Germany, 2002.

[11] Ramesh Nagappan, Robert Skoczylas, and Rima P. Sriganesh. *Developing Java Web Services*. John Wiley & Sons, 2003.

[12] David O'Riordan. Business process standards for web services. Technical report, Web Services Architect, 2002.

[13] Bhaskaran Raman, Sharad Agarwal, Yan Chen, Matthew Caesar, Weidong Cui, Per Johansson, Kevin Lai, Tal Lavian, Sridhar Machiraju, Z. Morley Mao, George Porter, Timothy Roscoe, and Mukun. The sahara model for service composition across multiple providers. In *International Conference on Pervasive Computing (Pervasive 2002)*, August 2002.

[14] Thomas Strang. *Service-Interoperabilität in Ubiquitous Computing Umgebungen*. PhD thesis, Ludwig-Maximilians-Universität München, October 2003.

[15] W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. web services - been there done that?, trends & controversies. *IEEE Intelligent Systems*, 18(1):72–76, February 2003.