Towards Realizing Global Scalability in Context-Aware Systems

Thomas Buchholz and Claudia Linnhoff-Popien

Ludwig-Maximilian-University Munich,
Mobile and Distributed Systems Group, Institute for Informatics,
Oettingenstr. 67, 80538 Munich, Germany
{thomas.buchholz, claudia.linnhoff-popien}@ifi.lmu.de
www.mobile.ifi.lmu.de

Abstract. Context-aware systems are systems that use context information to adapt their behavior or to customize the content they provide. Prior work in the area of context-aware systems focused on applications where context sources, Context-Aware Services (CASs), and users are in each other's spatial proximity. In such systems no scalability problem exists. However, other relevant CASs are subject to strong scalability problems due to the number of users, the geographical distribution of the context sources, the users, and the CAS as well as the number of organizations that participate in the provision of the context-aware system.

In this paper, we classify CASs according to their scalability needs and review context provision and service provision infrastructures with regard to their scalability. For building large-scale context-aware systems it is not sufficient to Zuse large-scale service provision and context provision infrastructures. Additionally an integration layer is needed that makes the heterogeneous access interfaces of the context provision infrastructures transparent for the CASs. We outline our proposal for such an integration layer and describe how it can be combined with an infrastructure that allows handhelds themselves to gather context information in their immediate vicinity via wireless technologies.

1 Introduction

Context-aware systems are systems that use context information to adapt their behavior or to customize the content they provide. In these systems three main types of entities can be distinguished: Context Sources (CS), Context-Aware Services (CASs) and users.

Early research in ubiquitous computing focused on applications where all entities involved in a user session are located in each other's spatial proximity like figure 1(a) illustrates. In such a setting there is no scalability problem. [1] coined the term "localized scalability" for this principle. Localized scalability is reached "by severely reducing interactions between distant entities" [1] or by locally restricting the distribution of information [2].

However, there are applications where the context sources, the CASs and the users are not colocated. Thus, interactions between distant entities are needed. In these situations the question arises how global scalability in context-aware systems can be reached. The key to this lies in replicating and distributing CASs as well as context information.

T. Strang and C. Linnhoff-Popien (Eds.): LoCA 2005, LNCS 3479, pp. 26–39, 2005.
© Springer-Verlag Berlin Heidelberg 2005

The paper is structured as follows: In section 2 we argue under which conditions a need for global scalability exists and give examples for applications. Section 3 reviews approaches for context provision as well as for service provision. In section 4 we discuss how the existing approaches can be combined and show which components are still missing in order to build large-scale context-aware systems. We describe our proposals for these components in section 4.1 and 4.2. In section 5 we outline directions for future research.

2 From Local Towards Global Scalability

In this section we will define classes of CASs with different scalability needs and will provide examples for these classes. To be able to discuss the scalability of the different application classes we first need to define what scalability is. According to [3] scale consists of a numerical, a geographical, and an administrative dimension. The numerical dimension refers to the number of users, deployed objects like e.g. context sources and services in a system. The geographical dimension means the distance between the farthest nodes, while the administrative dimension consists of the number of organizations that execute control over parts of the system. A large administrative dimension in general leads to heterogeneity. The various organizations are likely to use different hardware, software, protocols, and information models. In such an environment service interoperability becomes a problem. The interoperability problem is classically divided into a signature level, a protocol level, and a semantic level [4]. The syntax of the interface of a service is described on the signature level. It includes in general the name of any operation and the type and sequence of any parameter of the interface. On the protocol level the relative order in which methods of a service are called is specified. The problem of divergent understanding and interpretation is dealt with on the semantic level. [3] defines a system to be scalable if users, objects and services can be added, if it can be scattered over a larger area, and if the chain of value creation can be divided among more organizations without the system ”suffering a noticeable loss of performance or increase in administrative complexity.”

Locally scalable CASs (figure 1(a)): Most CASs that were built so far colocated the CAS, the context sources, and the users (see e.g. [5] for a survey). Typical examples are shopping and conference assistants. In these systems scalability is not an issue. All ways between the entities are short. The number of context sources and users is low. Moreover, the physical context sources were known when the CAS was developed. Thus, the CASs as well as the context sources were designed to interoperate.

CASs with heightened scalability needs (figure 1(b)-(d)): Recently, some CASs have been built where only two of the three types of entities are colocated and the third is far away. For example, many health surveillance applications [6] assume that sensors are attached to the body of persons with health problems. The gathered data is transmitted via the mobile Internet to a central database where it is analyzed by a physician or by a data-mining process that alerts a doctor if something is wrong (see figure 1(b)). Other applications colocate the user and the CAS (figure 1(c)), but receive context information from distant locations. An example are on-board navigation systems in cars that receive

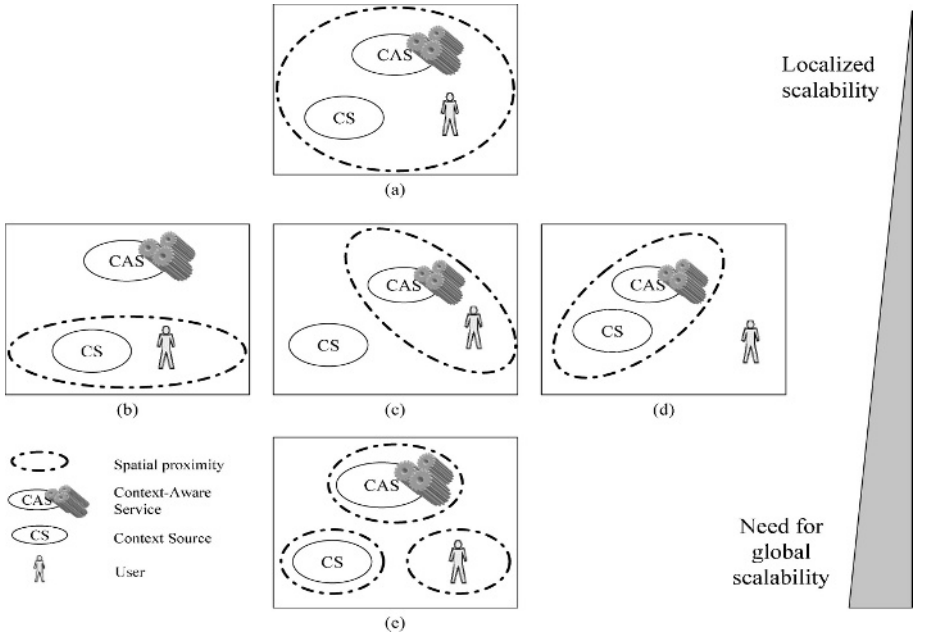

Fig. 1. Depending on the application context-aware services, context sources and users may be colocated or far apart

information about traffic jams via radio technologies. Surveillance of remote areas with the help of sensor networks is an application where in general the context sources and the CAS are colocated. When something interesting happens the user that is far away is informed (figure 1(d)). These types of CASs cause moderate scalability problems.

CASs with potentially global scalability needs (figure 1(e)): The largest scalability problems arise when the context sources, the CASs, and the users are far apart from each other. Examples are context-based push information services, contextualized yellow pages services, and server-based navigation applications. A context-based push information services for example is a friend-finder (cp. [7]). A friend-finder constantly tracks all members of a community within a large area. Whenever two members with similar interests enter each others vicinity, they are informed if their current activity does not forbid this (if e.g. one must not be disturbed because he is in a meeting). Another example is an application that reminds people to buy e.g. milk when they are near a supermarket. Contextualized yellow pages services (cp. [7]) help people to find nearby sights, restaurant, gas stations etc.. Scalability problems arise especially if dynamic properties of target objects (target context) and of objects between the user's current position and the target's position (transition context) are included into the recommendations given to the user. For example, a tour guide might incorporate the current waiting times at major sights into the proposed tour or a restaurant finder might suggest only places that can be reached within 15 minutes and still have seats available. To determine which restaurants are reachable within a certain time, the CAS finds out which

bus stops are near the user and when the next bus will arrive including all known current delays. If the user is within a car, the current traffic situation is taken into account. A server-based navigation system keeps all maps on the server and calculates routes based on the user's position, the point he wants to reach and dynamic information like traffic jams, weather condition, and road works.

The scalability problem of these applications is mainly caused by three properties of the overall system: 1. The CAS can be invoked from within a very large area. 2. Many persons might want to use the CAS. Thus, it must be ready to cope with a high workload. Moreover, the context of a plethora of entities must be accessible for the CAS. 3. The context sources will have different access interfaces in different regions and will be operated by different organizations. For example, a navigation system that uses traffic information for its recommendations will have to address that this information will be in a different language and format in Germany and France. Standards can partly solve this problem. However, many competing standards exist. For example, a traffic information system in the USA will express distances in miles and feet, while in Europe kilometers are used. For all these problems solutions must be found to allow for scalable context-aware services.

After having outlined for which classes of CASs global scalability is needed, we will review approaches for context provision and CAS provision with regard to their suitability for building globally scalable context-aware systems.

3 Approaches for Context Provision and CAS Provision

In order to build globally scalable context-aware systems the context provision process as well as the provision of the CAS must be scalable. We will address both subsystems in turn.

3.1 Scalable Context Provision

Scalability research in ubiquitous computing so far has mainly focused on the question how the context provision process can be made scalable. Figure 2 classifies the context provision approaches.

Context Provision approaches can be divided into infrastructureless and infrastructure-based approaches. If no infrastructure is employed, the CAS retrieves information from the context sources directly via wireless links or a multi-hop routing scheme is used. Multi-hoping is used in Wireless Sensor Networks (WSN) [8] where context sources collaborate and in Data Diffusion Systems [9, 10] where context sinks interact to distribute information. These technologies are only suitable if the distance between the context source and the context sink is not too large (limited geographical scalability).

Early infrastructure-based systems directly linked CASs with sensors that were attached to a Local-Area Network (tight coupling). To loosen the coupling two main approaches can be distinguished. Service-centric approaches provide a homogeneous interface for context retrieval from sensors. The Context Toolkit [11] (CTK) was the first representative of this school of thought. It encapsulates sensors in so called widgets that provide a unified interface to the CAS. The heterogeneity is diminished, but application-specific code still needs to interact directly with context sources, i.e. sen-

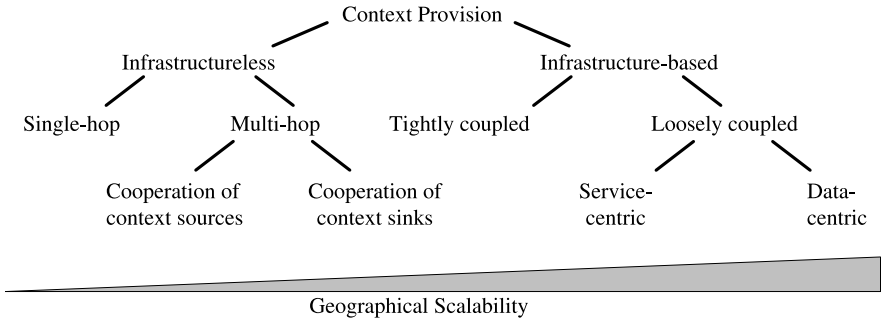

Fig. 2. Classification of context provision approaches

sors plus widgets. Other representatives are Solar [12] and iQueue [13, 14]. Data-centric systems introduced a data repository (e.g. a tuple-space) between the context sources and the CASs (e.g. [2]). The context sources provide context information to the data repository. CASs can query the repository via a query language. In these systems the CASs only need to comply to the service interface of the data repository and the used information model of the context sources.

Many proposals were made how geographically scalable context provision infrastructure can be built using the loose coupling paradigm, e.g. ConFab [7, 15], Solar [12], GLobal Smart Space (GLOSS) [16], Strathclyde Context Infrastructure (SCI) [17, 18, 19], IrisNet [20, 21], and iQueue [13, 14]. All these approaches connect the sensors to a local server that either acts as a data repository (ConFab, GLOSS, IrisNet), as an execution environment (Solar, iQueue), or as both (SCI). Some systems assign sensors to local servers merely based on geographical proximity (Solar, GLOSS, IrisNet, iQueue). Others make the data repositories a representative of entities in an environment (like people, places, and things) (ConFab, SCI). Data repositories are in general based on tuple-space technology (ConFab, GLOSS, SCI), but also XML-databases are employed (IrisNet). The large-scale context provision infrastructures must be able to answer queries that require data from many local servers. Thus, the local servers need to be confederated in some way. Many systems use Peer-to-Peer routing technology for this purpose (Solar, GLOSS, SCI). Others use XML hyperlinks between the data repositories (ConFab) or employ the Domain Name System (IrisNet). The information models that the approaches specify to express queries are very heterogeneous, though most of them are XML-based. Some describe context sources and chains of operators for the generation of context information. Others define functions that calculate the requested context information based on context data that is described in an attribute-based manner.

3.2 Scalable CAS Provision

Scalable context provision infrastructures are a very important building block for scalable context-aware systems. However, even if perfectly scalable context provision were

given, there would still be some challenges left to allow for completely scalable context-aware systems:

1. **Numerical dimension:** Large-scale CASs may have millions of users that collectively cause a workload that is much too high for a single server. Thus, replicating and distributing the CAS must be considered.
2. **Geographical dimension:** Large-scale CASs may have users that are spread over a very large area. In this case also relevant context sources will be strongly distributed. Since many interactions especially between the CAS and the context sources, but also between the user and the CAS will be needed, there is a potential that the long ways will lead to large response times of CASs. A solution is to distribute replicas of the CAS to servers that are close to the user and to the context sources.
3. **administrative dimension:** The large-scale context provision infrastructures possess heterogeneous access interfaces. Especially interoperability on the semantic level is a problem. Either a CAS must be able to deal with all of them or an infrastructure must be in place that makes the heterogeneity transparent for the CAS.

Little work has been done on the question how scalable CASs can be built. The only works we are aware of are [6], [22], and [23]. These works suggest to expose context sources and context repositories to the system as grid services. A grid service is a Web Service that conforms to a set of conventions [24]. By using grid technology interoperability on the signature level is reached and the scalability properties of grids can be reused. Besides grids other technologies for scalable service provision exist. They are classified in figure 3.

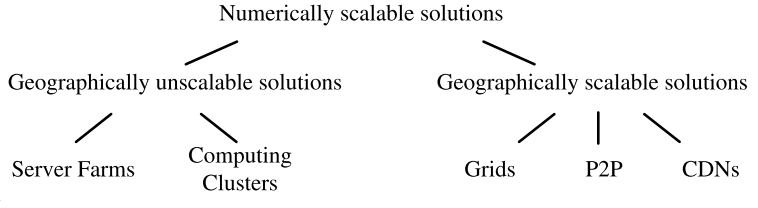

Fig. 3. Classification of CAS provision approaches

If too many requests need to be answered by a web-based application, one of the most often used approaches is to employ a server farm. The application is replicated and installed on many servers that are interconnected by a common high-speed Local-Area Network (LAN). The workload is evenly distributed among the servers by an intelligent switch. Computing clusters are similar to server farms. Computing problems that are too large for a single server are divided into many parts that are solved in parallel by many CPUs. Since the execution of one session of a CAS in general does not overcharge a single server, server farms are more suitable for CASs than computing clusters. Server farms address the numerical dimension, but the geographical dimension of the scalability remains unsolved.

Infrastructures that solve the numerical and geographical scalability problem are grids [25], peer-to-peer (P2P) networks [26], and Content Delivery Networks (CDNs) [27]. All these technologies are able to coordinate globally distributed servers. All three infrastructures have their specific strengths. Grids excel at monitoring resources and balancing load. P2P networks perform content-based routing in a very scalable manner and CDNs are especially good at assigning client requests to nearby servers. The three technologies are converging. Grids will use P2P routing. Recent proposals for P2P networks allow for routing to nearby servers and CDNs use the Apache Tomcat container like the Globus Toolkit [28] (the most widely used toolkit for grid technology) as an execution environment for applications.

To build globally scalable context-aware systems the best choice is to use one of these three numerically and geographically scalable infrastructures to dynamically distribute replicas of CASs and to combine it especially with large-scale context provision infrastructures. However, coupling the CAS provision infrastructures with the various context provision systems poses some challenges like we will discuss in the next section.

4 Components for Scalable Context-Aware Systems

In order to build globally scalable context-aware systems it is necessary to dynamically replicate and distribute CASs on the servers of one of the three numerically and geographically scalable service provision infrastructures that were outlined in the last section. This infrastructure forms the upper layer in figure 4. In this way the numerical and geographical scalability of the CAS is assured.

The replicas of the CASs that are placed on the geographically distributed servers retrieve context information from regional Context Information Services (CIS) (see lower layer in figure 4). A CIS is an abstraction. It is any service that provides context information. In most cases it will be a large-scale context provision infrastructure, but it can also be a single sensor or a user's handheld that provides context information. By combining a large-scale service provision infrastructure with large-scale context provision infrastructures numerical and geographical scalability is reached.

However, the administrative dimension is still a problem: The CISs in the various regions may possess different access interfaces, even if they provide the same type of context information. Figure 4 illustrates this with diversely shaped symbols. Since the developer of a CAS does not know at design time on which servers his CAS will be deployed and which CISs will be used, he cannot anticipate what the access interface of needed CISs will be like. This is a major problem. The replicas of a CAS need an execution environment that is the same on every server. This means that also the access to context information must be identical on every server. Thus, the CAS distribution infrastructure needs to provide an integration layer that binds to the heterogeneous access interfaces of CISs, maps the retrieved information to a standard information model, and provides it to the CASs in a unified way. Such an integration layer is our CoCo infrastructure. It will be outlined in section 4.1.

The CoCo infrastructure resides on each server and binds to local or regional CISs. In general this CISs are large-scale context provision infrastructure. However, for some

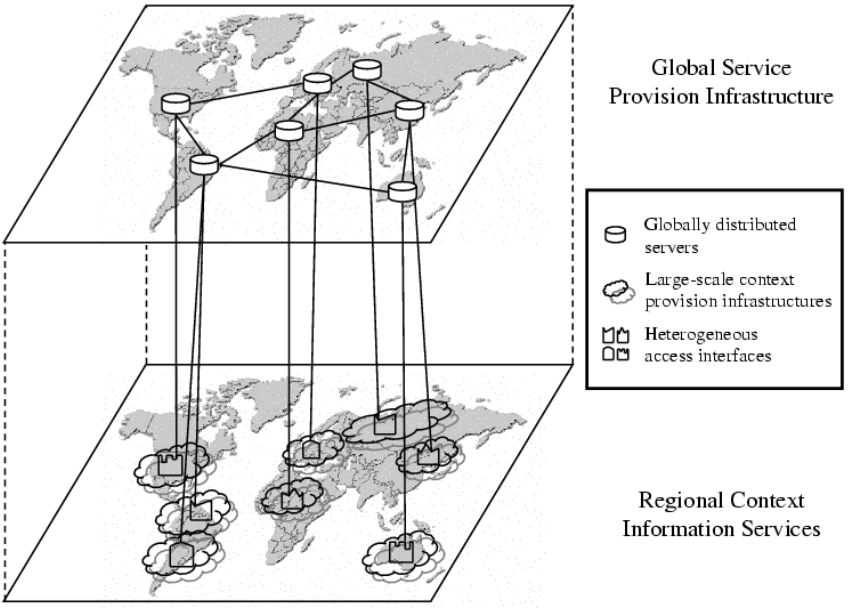

Fig. 4. Combining a scalable service provision infrastructure with large-scale context provision infrastructures provides numerical and geographical scalability. However, the administrative dimension of the scalability problem is yet to be solved

applications it might also be useful that the user's handheld itself looks for context information in its immediate vicinity via wireless technologies and passes the context information to the CoCo infrastructure when it invokes the CAS via e.g. UMTS or GPRS. For example, somebody might be looking for restaurants that can be reached within 15 minutes with public transportation means and that are not overly crowded. The information which bus stations are near the user is probably most efficiently retrieved via wireless multi-hoping methods. To find out which candidate restaurants still have free seats available, however, is an information that is generated too far away from the user to be accessible wirelessly. For the wireless gathering of context information in the user's immediate vicinity we developed an infrastructure that will be described in section 4.2.

4.1 Context Composition Infrastructure

The Context Composition (CoCo) infrastructure [29] is a set of collaborating components that generate requested context information. It solves the following problems: It binds to regional CISs and can translate the provided context information into the standard information model. Moreover, it is able to execute workflows that are needed to generate higher-level context information from low-level context information. These workflows and needed pieces of context information are specified in a CoCo document using the CoCo-language which is based on XML. It can be represented as a graph.

Figure 5 shows an example for the generation of context information that a restaurant finder might need. CoCo documents are mainly composed of factory nodes that specify which pieces of context information are needed and operator nodes that describe how context information needs to be adapted, selected, aggregated, or processed in any other way. A directed edge between two nodes stands for a context information flow. In the example the user is looking for nearby restaurants or beer gardens (if the weather is fine) that match his preferences he specified in a profile. He invokes the CAS and passes his phone number. The CAS dynamically adds this phone number to the CoCo document that describes the general context information needs of the restaurant finder. This document is given to the CoCo infrastructure. Based on the phone number the user's profile and his current location can be found out. Not until the user's position has been retrieved, the temperature and the likelihood that it will rain can be requested because these pieces of context information are dependent on the location. Since the CAS is not interested in the temperature and the likelihood of rain directly, an operator node is invoked that uses the former information to determine whether the weather can be considered as good. The retrieved user preferences, the user's location and the decision if the weather is good are given back to the CAS.

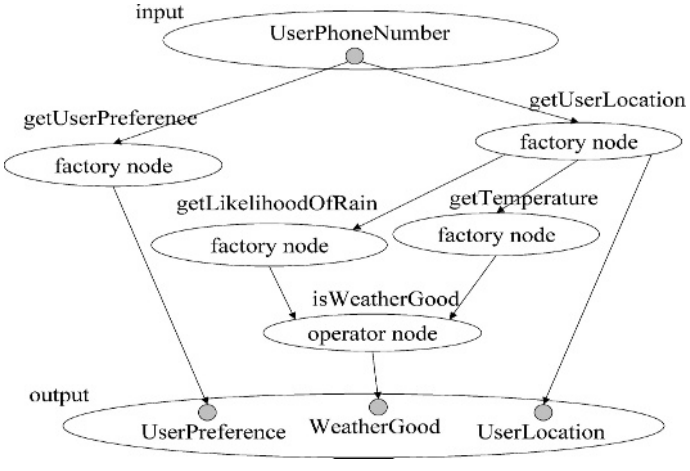

Fig. 5. Example of a CoCo-Graph

The infrastructure that executes the CoCo-Graph consists mainly of a CoCo-Graph Controller, a Context Retriever, the CoCo Processor and two caches like depicted in figure 6.

The CoCo-Graph Controller receives the CoCo document from the CAS and executes the specified steps. Thus, the CoCo-Graph Controller is the component that is in charge of flow control. Whenever it parses a factory node, it sends the corresponding information to the Context Retriever that responds with the respective context information. Operator nodes result in a call to the CoCo Processor. When the output node is reached, the results of the execution of the CoCo document are returned to the CAS.

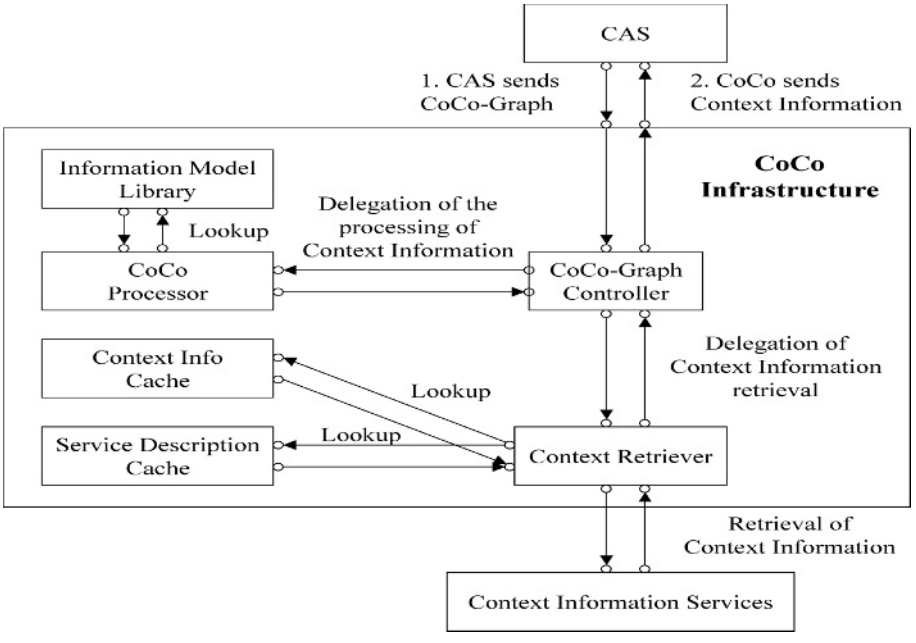

Fig. 6. CoCo Infrastructure

The Context Retriever discovers CISs, selects the most appropriate one and returns the retrieved context information to the CoCo-Graph Controller. Both, the descriptions of discovered CISs and the context information, 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 many other pieces of context information. For example, the CoCo Processor would process the operator node ”isWeatherGood” to decide if the weather is good or bad, based on the retrieved temperature and the likelihood of rain. The CoCo-Graph Controller also delegates implicit instructions to the processor. These are conversion operations between different scales, i.e. formats, of the same context information item. In our scenario, the CoCo Infrastructure locates the user and passes his position to the factory node needed to retrieve the temperature at the user‘s position. The position might be expressed in WGS84-coordinates, while there might be only CISs available that accept coordinates exclusively in UTM-format. At this point, the CoCo-Graph Controller invokes the CoCo Processor to try to convert the information. For its task the CoCo Processor needs semantic information about the information model and requires conversion operations between the various scales of the same types of context information. This knowledge and functionality is stored in the Information Model Library.

The CoCo infrastructure was conceived as an integration layer between large-scale context provision infrastructures and execution environments for CASs. In the next section we will describe how the CoCo infrastructure can be combined with wireless context provision methods.

4.2 Context Diffusion Infrastructure

We assume that when a user subscribes to a CAS, he downloads a thin client to his handheld. This thin client uses a Context Diffusion Infrastructure that is already installed on the handheld to gather context information. The main principle of Context Diffusion is that static server nodes announce information via local wireless radio technologies like Bluetooth or IEEE 802.11. Mobile nodes listen to this information and cooperate to disseminate the information by word of mouth.

The core of our Context Diffusion Infrastructure is our data diffusion protocol [30]. It is executed whenever two nodes come into radio range. These can either be two mobile nodes or an immobile and a mobile node. The protocol works as follows:

1. First, profiles are transmitted between the peers. The profiles specify in which types of context information the profile owner is interested.
2. The peer's profile is used to search through the handheld's cache of formerly obtained pieces of context information.
3. All cached pieces of context information that might be of interest for the exchange partner according to his profile are transmitted and are then stored on the receiver's side.

Since each handheld frequently executes a cache invalidation scheme that deletes all pieces of context information that have become outdated or invalid because the user has moved out of the region where the specific piece of context information could be useful, only valid information that refers to the current region is exchanged.

The Context Diffusion Infrastructure offers an API for thin clients to add entries to the profile. Depending on the application's collection strategy, it can either add its profile during its installation phase or when it is invoked. In the former case, the handheld scans continuously its environment for available context information, in the latter case, context information collection is initiated on demand. The infrastructure allows for push notification and for a request mode. Push notification means that the thin client is informed everytime an interesting piece of context information is received. Alternatively, the thin client can search the cache for relevant entries using the request mode. Our favorite configuration is that the thin client initiates the context retrieval when it is invoked. After a fixed amount of time it searches through the cache for relevant context information, invokes the backend of the CAS via GPRS or UMTS and transfers the context information to it.

Whenever the backend of a CAS is invoked by a client, it analyzes the request to see whether context information was passed. If this is the case, the provided context information is given to the CoCo infrastructure. The CoCo infrastructure receives the information, transforms it into the standard information model if needed, and writes it into the context information cache. The CAS operates in the normal way and requests context information from the CoCo infrastructure with a CoCo document. Since the Context Retriever within the CoCo infrastructure always starts looking in the cache whether needed context information is already stored, the context information provided by the client is very naturally introduced into the standard process of CoCo. The functionality that is in place to check whether cached context information is usable for a certain problem is reused to validate context information that was passed by the client.

Additionally, the information that was gathered by one client is potentially reusable for others. By writing the context information that was gathered by the client into the CoCo cache instead of directly using it, it becomes transparent for the CAS in which way context information was retrieved.

5 Conclusion and Future Work

In this article we have shown that some Context-Aware Services (CASs) need to be globally scalable. We have outlined approaches for context provision and CAS provision and evaluated them with regard to their scalability. Good candidates as a suitable CAS provision infrastructure are grids, P2P networks, and CDNs. These systems need to be coupled with large-scale context provision infrastructures. To provide a homogeneous access interface to context information for replicas of CASs the CAS provision infrastructures need to provide an integration layer that makes the heterogeneity of the access interfaces of the context information services transparent for the CASs. The CoCo infrastructure is such an integration layer. It can be combined with an infrastructure that allows to gather context information in the immediate vicinity of a user. These approaches allow to build globally scalable context-aware services.

Currently, we are working on the development of efficient, but easily applicable heuristics for the distribution of CASs in CDNs. We are developing a simulation that allows to evaluate the resource consumption and improvements of user-perceived latencies that are incurred by the various heuristics. Furthermore, we are refining our infrastructures. Especially, the development of a generic model for context information remains one of the hard research questions.

References

1. Satyanarayanan, M.: Pervasive Computing: Vision and Challenges. IEEE Personal Communications (2001)
2. Schmidt, A.: Ubiquitous Computing - Computing in Context. PhD thesis, Computing Department, Lancaster University, England, U.K. (2002)
3. Neuman, B.C.: Scale in distributed systems. In: Readings in Distributed Computing Systems. IEEE Computer Society, Los Alamitos, CA (1994) 463–489
4. Strang, T., Linnhoff-Popien, C.: Service Interoperability on Context Level in Ubiquitous Computing Environments. In: Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w), L'Aquila/Italy (2003)
5. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College (2000)
6. Barratt et al., C.: Extending the Grid to Support Remote Medical Monitoring. In: 2nd UK eScience All hands meeting, Nottingham, U.K. (2003)
7. Hong, J.I., Landay, J.: An architecture for privacy-sensitive ubiquitous computing. In: Proceedings of The Second International Conference on Mobile Systems, Applications, and Services (Mobisys 2004), Boston, MA, USA (2004)
8. Al-Karaki, J.N., Kamal, A.E.: Routing techniques in wireless sensor networks: A survey. IEEE Wireless Communications (2004)

9. Papadopouli, M., Schulzrinne, H.: Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In: ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc), Long Beach, California (2001)
10. Becker, C., Bauer, M., Hähner, J.: Usenet-on-the-fly: supporting locality of information in spontaneous networking environments. In: CSCW 2002 Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments, New Orleans, USA (2002)
11. Dey, A.: Architectural Support for Building Context-Aware Applications. PhD thesis, College of Computing, Georgia Institute of Technology (2000)
12. Chen, G.: SOLAR: Building a context fusion network for pervasive computing. PhD thesis, Dartmouth College, Hanover, New Hampshire, USA (2004) Technical Report TR2004-514.
13. Cohen, N.H., Purakayastha, A., Wong, L., Yeh, D.L.: iqueue: a pervasive data-composition framework. In: 3rd International Conference on Mobile Data Management, Singapore (2002) 146–153
14. Cohen, N.H., Lei, H., Castro, P., II, J.S.D., Purakayastha, A.: Composing pervasive data using iql. In: 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002), Callicoon, New York (2002) 94–104
15. Heer, J., Newberger, A., Beckmann, C., Hong, J.I.: liquid: Context-aware distributed queries. In: Proceedings of Fifth International Conference on Ubiquitous Computing: Ubicomp 2003, Seattle, WA, USA, Springer-Verlag (2003) 140–148
16. Dearle, A., Kirby, G., Morrison, R., McCarthy, A., Mullen, K., Yang, Y., Connor, R., Welen, P., Wilson, A.: Architectural Support for Global Smart Spaces. In: Proceedings of the 4th International Conference on Mobile Data Management. Volume 2574 of Lecture Notes in Computer Science., Springer (2003) 153–164
17. Glassey, R., Stevenson, G., Richmond, M., Nixon, P., Terzis, S., Wang, F., Ferguson, R.I.: Towards a Middleware for Generalised Context Management. In: First International Workshop on Middleware for Pervasive and Ad Hoc Computing, Middleware 2003. (2003)
18. Stevenson, G., Nixon, P.A., Ferguson, R.I.: A general purpose programming framework for ubiquitous computing environments. In: Ubisys: System Support for Ubiquitous Computing Workshop at UbiComp 2003, Seattle, Washington, USA (2003)
19. Thomson, G., Richmond, M., Terzis, S., Nixon, P.: An Approach to Dynamic Context Discovery and Composition. In: Proceedings of UbiSys 03: System Support for Ubiquitous Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing (UbiComp 2003), Seattle, Washington, USA (2003)
20. Gibbons, P.B., Karp, B., Ke, Y., Nath, S., Seshan, S.: IrisNet: An Architecture for a World-Wide Sensor Web. IEEE Pervasive Computing **2** (2003)
21. Deshpande, A., Nath, S., Gibbons, P.B., Seshan, S.: Cache-and-query for wide area sensor databases. In: ACM SIGMOD 2003. (2003)
22. Storz, O., Friday, A., Davies, N.: Towards 'Ubiquitous' Ubiquitous Computing: an alliance with the Grid. In: System Support for Ubiquitous Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing (UbiComp 2003), Seattle (2003)
23. Jean, K., Galis, A., Tan, A.: Context-Aware GRID Services: Issues and Approaches. In: International Conference on Computational Science, Krakow, Poland (2004) 166–173
24. Tuecke et al., S.: Open Grid Services Infrastructure (OGSI), Version 1.0. Global Grid Forum Draft Recommendation (2003)
25. Berman, F., Fox, G., Hey, A.J., eds.: Grid Computing: Making the Global Infrastructure a Reality. Wiley (2003)
26. Lua, E.K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. IEEE communications survey and tutorial (2004)

27. Rabinovich, M., Spatscheck, O.: Web Caching and Replication. Addison Wesley (2002)
28. Globus Alliance: The Globus Toolkit (2004) http://www-unix.globus.org/toolkit/.
29. Buchholz, T., Krause, M., Linnhoff-Popien, C., Schiffers, M.: CoCo: Dynamic Composition of Context Information. In: The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2004), Boston, Massachusetts, USA (2004)
30. Buchholz, T., Hochstatter, I., Treu, G.: Profile-based Data Diffusion in Mobile Environments. In: Proceedings of the 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2004), Fort Lauderdale, USA (2004)