Distributed Context Data Management

Fredrik Kilander[1], Wei Li[1], Carl Gustaf Jansson[1], Theo Kanter[2], Gerald Maguire[3]

[1] Dept. of Computer and Systems Sciences, KTH and Stockholm University, Forum 100, SE-164 40 KISTA, Sweden
[2] Ericsson Research, SE-164 80, Kista, Sweden
[3] The Royal Institute of Technology, Forum 100, SE-164 40 KISTA, Sweden

Abstract. We argue that general context data distribution can be achieved by establishing a network of *context managers*. The context managers have a uniform core which is surrounded by the appropriate functionality to make it deployable in applications, user and organizational infrastructure, and on top of sensors. When access to the context manager is mediated and policed, we say that it is a *context service*, a publicly recognisable, addressable and tangible entity. Clients that wish to consult the context manager for updated context information, must turn to the service interface where they may face rejection for reasons of security, personal integrity or workload.

1 Introduction

The Adaptive & Context-Aware Services (ACAS) research project[4] focuses on distributed collaborative work environments. This involves the utilization of IT artifacts (devices, terminals) that dissolve into small components that increasingly melt into the fabric of everyday things, "smart dust" being an extreme case. Ad-hoc functionality on all systems levels implies dynamic and automatic configuration of public and private artifacts and services. Both the technical solutions as well as possible business models are of importance.

The ACAS project has as its main goals:

- to achieve a better utilization of available resources (devices, services, and communication technologies) in users' work situations and
- to achieve a transparent and personalized work situation for the users given available resources.

Both objectives provide strong support for the overall objective to create affordable wireless infrastructure and services. In the first case, better utilization of resources leads to considerable cost savings in infrastructure. In the second case, the ability to create seamless and adapted work situations for mobile workers result in operational cost savings for collaborative work in any physically distributed organization.

[4] `http://psi.verkstad.net/acas`

Three main research areas have been identified: *Ad hoc service environments*, focusing on automatic compositions of artifacts and services given the purpose of the application and the user's context. *Seamless adaptive services* involve adaptation of services taking into account the properties of available infrastructure resources and given the purpose of the application and user's context. *Smart adaptive infrastructure* is concerned with adaptation of infrastructure resources involved in service delivery given the purpose of the application and the user's context.

In the project's first two and a half years, we have designed a first architecture providing a network for the publication, aggregation, and dissemination of group state information. The service architecture was based on SIP, and the proposed SIMPLE presence publication mechanism [10, 4]. We have also considered

- Context acquisition, refinement and distribution (management)
- Context modelling
- Service discovery, allocation and composition
- Privacy aspects

In this paper we argue that general context data distribution can be achieved by establishing a network of *context managers*. The context managers have a uniform core which is surrounded by the appropriate functionality to make it deployable in applications, user and organizational infrastructure, and on top of sensors. Context information in the network is likewise expressed in a common form to facilitate rule-guided selection, translation, and inference.

When access to the context manager is mediated and policed, we say that it is a *context service*, a publicly recognisable, addressable and tangible entity. Clients that wish to consult the context manager for updated context information, must turn to the service interface where they may face rejection for reasons of security, personal integrity or workload [9].

We envision context managers to appear in three places:

- Embedded in applications, as the implementation of a context-sensitive API.
- In general context services that represent a user, an organization, a location or some other conceptual entity.
- As sensor services that expose sensor data in a common and exchangable format.

Taken together, the applications, context and sensor services forms a context information network. By implementing context managers as services, ordinary service discovery techniques can be employed to establish connections in the network. In addition, proximity detection can be used to establish ad-hoc relationships, such as between a user's laptop and the room it is in.

2 Context Managers

Context information originates in sensors, it is made available by publishing, and reaches clients through subscriptions. The gap between fundamental sensor

information and the abstract relationships convenient to end-user applications is bridged by context refinement. In this section we briefly consider what we believe is of importance to real-world implementations.

We assume that context-sensitive support in an application is a rather small part when compared to the application's main purpose. Adaptation and context-based response can give a powerful addition to the standard behaviour of the application, but has in general no intrinsic value. It is also desirable that the effort of extending an application with context awareness is small.

Another consideration is that many different applications are likely to benefit from the same context information. Location, for example, is a powerful concept that can be exploited in many ways. Thus it is likely that several applications executing for the benefit of a particular user, possibly on the user's laptop or wearable device, can utilise the same location information and hence share subscriptions.

We would like to hypothesize that end-point applications in a context information network are best served by subscribing to a minimum of information. In practice this means that they only receive context data elements with abstractions that are directly and immediately relevant to the application. This simplifies the matching process in the application, as it reduces the size of the state description that must be maintained, and (for battery-powered and wireless units) it reduces the amount of information that must be exchanged across the network.

Our approach is to avoid duplication of effort and to move context refinement out of the applications and into the context data network. This is achieved by making the context manager programmable by its clients. In the most relaxed scenario, applications pass a set of context rules (originally provided by the application developer) to their context API and are then notified whenever the state description generated by the output of the rules is updated.

Context-sensitive applications on a laptop or PDA would benefit from an onboard context service. If the the context managers embedded in the applications can pass on the subscription requests to a device-specific context service, they will each require less CPU and memory in order to serve their applications. The device context service in turn will be able to collapse subscriptions for common context information (e.g. location) and thus reduce network traffic.

Similarly, context refinement that is needed by several applications but has limited scope[5] can be shared by having the context manager govern the refiner processes. This identifies the location of context refiners that are part of the middleware for a system of applications, and it makes them easier to design and deploy.

[5] By limited scope we mean that the refinement is useful only to the application owner, and not in general.

3 Context servers

The concept of the context server has matured rather slowly with us. For a long time we thought of it as a *personal server*, because we were concerned with privacy issues. The idea of a context server is, however, general enough to allow for the same functional solution to be deployed for *different* audiences, just as with www URLs; some are personal, others are public, and yet others only cater to special groups of clients.

The context server forms an outer shell around the context manager, by policing access to the information within. This is necessary because there may be situations when the owner of the context server is willing and ready to share parts of the context repository with certain other clients. These subscription requests must be evaluated, accepted or modified before they are allowed into the context manager. For example, the user may be willing to share fine-grained location information with personal friends, coarse-grained location with an employer or customers, and reveal no location at all to anyone else.

The context service aggregates, refines or creates context information for one or more subscribers. In general, the context manager has a service interface, a refinement and subscription module, and a client role. Remote subscribers announce themselves and their needs on the service interface. The CM splits the subscription into two parts, one which it believes it can answer locally, and the rest which must be obtained by resubscribing at other context managers, in its client role. When updated context information becomes available, the context manager notifies each subscriber according to the local part of the subscription.

In our view there are three distinct and vertically ordered classes of context managers:

The application context manager (figure 1) is loaded or consulted by an application in order to serve that application with context information. In this instance, the context manager's service interface is likely to appear as a separate thread of execution, controlled by a set of functions in the application's address space, rather than as a remote service, although that view is not prohibited. Likewise, notifications from the CM are implemented as callbacks to functions specified by the application, again within the application's address space.

The general context manager (figure 2 left) appears as a stand-alone service, running either in a single instance on a small device or on the behalf of a user or organisational element on a persistent and connected server computer. The general context service implements the full remote service interface and client role, as well as the capacity to host multiple subscribers.

The sensor context manager (figure 2 right) effects the transition from sensor data to the context description languages in use. This context manager has the full service interface and subscription machinery, while the client role is replaced by access mechanisms to whatever sensors the CM supports.

Typically, each context-sensitive user-level application would be equipped with an instance of the application context manager. Then, each of the user's

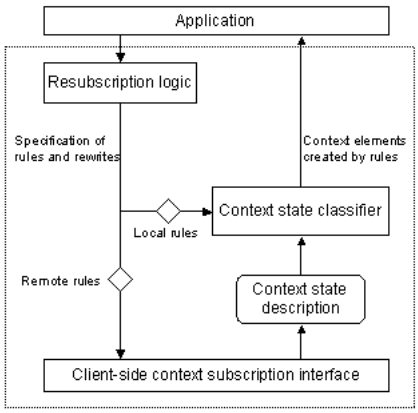

Fig. 1. Application Context API

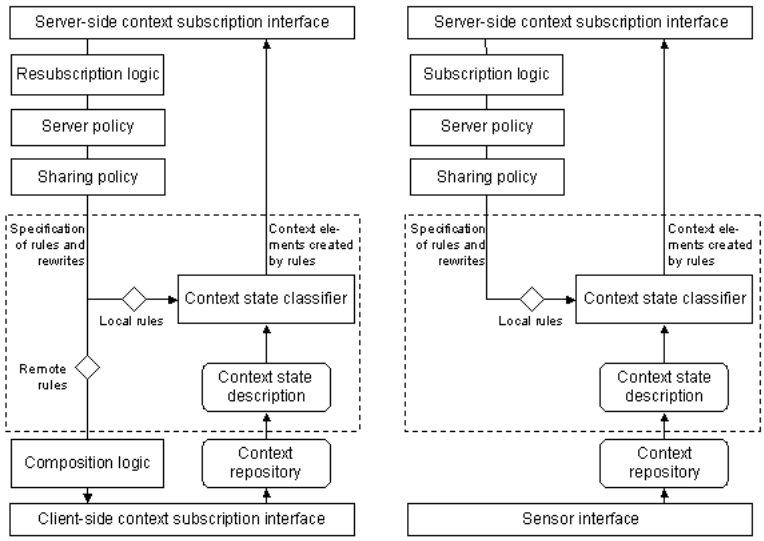

Fig. 2. General (left) and Sensor (right) Context Service

computers (PDA, laptop, PC) has a general context manager local to the device. It consults a persistently installed general context manager dedicated to the user. The user's personal and general CM is distinguished, not because of its technology but due to its role. Among its peers are the personal context managers of other users, and those that represents parts of the environment (rooms, stores) or an organization.

Distributed out in the infrastructure, running on whatever hardware is appropriate, are sensor context managers. These provide the actual measurements.

Access to context information is controlled by a set of policies installed in the context server. We do not specify how these are to be defined, expressed, or implemented; nor do we dictate how clients identify themselves or the mechanism used to authenticate them. All that is future work.

3.1 Proximity-based Discovery

As for a mobile user, we anticipate certain interesting scenarios in which the user moves in and out of different environments where support for context information exists. Detectors are mounted in fixed locations or worn by mobile users. As two detectors make contact, both note the identity of the other (MAC address or similar) and responds with a URI which is the address of its own context service. The two parties, user and room, are now aware of each other and can receive what context information each sees fit to expose.

4 The Context Information Network

The network consists of context managers on three separate levels:

Application level The level closest to the applications are the end consumers of context information. Applications typically are not interested in low-level sensor data; instead they prefer to react to particular abstractions like "Is there a friend close right now?"

General level The level where context is distributed and abstractions of a general nature are created.

Sensor level The level closest to real or virtual sensors. This level transcribes or interprets sensor readings into context information languages that can be made subject to generalisation by proper reasoning.

So far we have not addressed the general problem of how context managers are able to find other context managers to which they need to subscribe. We only have some initial ideas for this, so for the moment let it suffice to specify some of the properties requested of a solution.

We believe that context managers will be able to automatically formulate subscription requests for context information by analysing the rules given to them by applications. In order to place these subscriptions with useful sources (other context services), they must have some notion of where to send them. We have indicated how proximity-based detection and URI exchange is one way to

find this information. Another way is to use static configurations and long-term relationships with trusted parties who always subscribe to the same information.

We have entertained the idea that context services could be built as a peer-to-peer network. Even though the policies of the context service are designed to restrict uninhibited data exchange, there is no advantage in preventing the sharing of context data when the party generating it wants to do so. This means that peer-to-peer strategies must be built upon long-term relationships, or the existence of searchable directories over public context information and context information sources. An example of friend of a friend information propagation is described in Diego Delgado's thesis [6].

5 Context Refinement

Context refinement and subscription are unified in the Tryton rule language, an experimental production language developed by us for this purpose. Essentially, when a client wish to subscribe to context, it composes a Tryton program in which production rules imply the necessary context information. The left-hand side of each rule requires the corresponding context information to be available in the context manager, while the right-hand side specifies the context element that is to be sent to the client.

The context manager must examine the left-hand side of each rule and adjunct it with the following properties:

1. the context relation is unknown and must be subscribed for elsewhere
2. the context relation is local to the rule set (an abstraction produced by the rhs of other rules in the subscription)
3. the context relation is known and can be provided locally

For case 1, the CM would turn to other context managers that it has knowledge of and request subscriptions[6].

6 Related Work

There have been numerous attempts to create context-aware systems to support personal mobility and context awareness. We will only address those which are most relevant to our work. A number of earlier architectures [1, 12] share the concept of a "Directory Service" which keeps user profiles, lists user's devices, and specifies what services are best suited for use in different situations. These architectures can provide some support of user mobility in terms of supporting switching between multiple communication devices. However, this directory service model relies on static information predefined by the user, and does not consider the user's changing context, nor does it support the uses of devices and

[6] To prevent message feedback loops the resubscription history, hop count and hop limit must be present in the subscription request.

services located in remote places or newly deployed in familiar places which the user may not know of in advance.

K. El-khatib et al. [8] used Personal Agent (also implemented on SIP) to determine how to render mobile user's incoming calls in ubiquitous computing environment with support of better performance and interaction means according to user's profile and available services. Stefan Beger and Henning Schulzrinne et al. have elaborated comprehensively in a recent paper [3] on how to construct ubiquitous computing system using SIP together with many other standard protocols. We agree with them that a global-scale ubiquitous computing system should be divided into different domains, and through the SIP servers in those domains, the user can utilize the rich resources in the visited domains. However, except for many similarities due to the use of common technologies and protocols such as SIP and Bluetooth, there are remarkable differences to distinguish our work: first, we emphasize how the local infrastructure instead of the user's mobile device delivers context data back to the user's personal context service; secondly we have introduced a context refiner concept to infer high-level context information.

For some additional related work see the Aura project at Carnegie Mellon University [2].

6.1 The Solar System

The Solar system [5] is a prototype implementation of a graph-based abstraction for context collecting, aggregation, and dissemination. Sources (sensors) generate events that flow through a directed acyclic graph, passing one or more operators and is finally delivered to a subscribing application. Operators may act as filters, transformers, aggregators, and mergers, to allow context refining and modification in order to deliver meaningful data to the subscribing application (format, level of abstraction, etc.). The operators may participate in delivering context to many other operators and/or subscribing applications, and hence they may be reused in order to minimize the work and communication needed to disseminate context information.

6.2 The Context Toolkit

The Context Toolkit, [11, 7] developed at the Georgia Institute of Technology (Georgia Tech)[7], supports the development of context-aware applications using *context widgets*; software components that provide context information to applications. The widgets makes it possible to incorporate sensor information in applications in a manner similar to the way a modern GUI is assembled from basic elements. Widgets can be of different types and have different attributes. Applications can register with a widget that will trigger callbacks when changes

[7] With the move of Anind K. Dey from (Georgia Tech) to the University of California, Berkeley; the latest information can be found at: http://www.cs.berkeley.edu/ ~dey/context.html

occur. Widgets have several components with different responsibilities. The lowest level interfaces to a physical sensor, and hides the lower level details of how to control the specific sensor. The middle layer is concerned with abstracting data and combining data from the lower level. The highest level coordinates the underlying components and provides the callback interface to applications.

One advantage of the layered design is that it clearly separates low-level details of sensor hardware and sensor values, from higher level context information interesting to applications. Parts of a widget can be upgraded or replaced without affecting the applications.

The Context Toolkit provides the means for the aggregation of data from different sensors, as well as support for translation of low-level sensor data to high-level data that can be used by applications directly. The system does not have an explicit infrastructure approach but rather supports the creation of standalone applications. The Context Toolkit provides similar functionality as the ACAS system, such as context queries and refinement of context data, but it lacks the ability to migrate some of the processing into the infrastructure, which the ACAS system permits.

6.3 WASP

The Web Architectures for Service Platforms (WASP) [13] is developed at the University of Twente, Enschede, in the Netherlands. The platforms are designed to support context-aware applications specifically in the 3G environment using Web Service technologies. The goal is to support deployment of a large range of context-aware applications still unanticipated. This is enabled using WSL (WASP Subscription Language), which they have designed to be used to communicate with their platform. The platform connects context-aware applications with context providers (sensors) and third party service providers, it allows the applications to place queries and specify the ordering of actions to occur, at the platform. The platform itself handles the collection and aggregation of context as well as executing the specified actions.

7 Concluding remarks

The infrastructure proposed by the ACAS project offers the tantalizing possibility of a generally applicable structure that recursively reapplies itself from applications, through context services down to sensors. However, the deceptive simplicity of this three-tiered hierarchy raises several additional issues. For example, what methods of data transport should be used? Is the rule-based context refinement strategy practical or will the processing delays be intolerable? We are currently working to answer some of these questions through practical implementations and simulations.

References

1. N. Anerousis, R. Gopalakrishnan, C. Kalmanek, A. Kaplan, W. Marshall, P. Mishra, P. Onufryk, K. Ramakrishnan, and C. Sreenan. Tops: An architecture for telephony over packet networks. *IEEE Journal on Selected Areas in Communications*, 17(1):91–108, 1999.
2. Project aura, distraction-free ubiqutious computing. http://www-2.cs.cmu.edu/~aura/.
3. Stefan Berger, Henning Schulzrinne, Stylianos Sidiroglou, and Xiaotao Wu. Ubiquitous computing using sip. In *Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'2003)*, pages 82–89. ACM Press, 2003. ISBN:1-58113-694-3.
4. B. Campbell, S. Olson, J. Peterson, J. Rosenberg, and B. Stucker. Simple presence publication mechanism. http://www.ietf.org/internet-drafts/draft-ietf-simple-publish-00.txt, February 2003. Work in progress. Expires August 25, 2003.
5. G. Chen and D. Kotz. Solar: A pervasive-computing infrastructure for context-aware mobile applications. Technical report, Department of Computer Science, Dartmouth College, Hanover, NH, USA., February 2002.
6. Diego U. Delgado. Implementation and evaluation of the service peer discovery protocol. Master's thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, to appear, May 2004. Draft available at: http://www.imit.kth.se/ ~awd/ publications/ 040510_Diego_UrdialesDRAFT.pdf.
7. Anind K. Dey, Daniel Salber, and Gregory D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCI) Journal*, 16(2–4):97–166, 2001. Anchor article of a special issue on context-aware computing.
8. K. El-Khatib, N. Hadibi, and G. v. Bochmann. Support for personal and service mobility in ubiquitous computing environments. Technical report, School of Information Technology & Engineering, University of Ottawa, 161 Louis Pasteur St., Ottawa, Ont., K1N 6N5, Canada, 2003. http://beethoven.site.uottawa.ca/dsrg/ PublicDocuments/ Publications/ ElKh03a.pdf (11-May-2004).
9. Wei Li, Martin Jonsson, Fredrik Kilander, and Carl Gustaf Jansson. Building infrastructure support for ubiquitous context-aware systems. *Lecture Notes in Computer Science, Springer-Verlag GmbH*, 3358/2004:509–518, November 2004.
10. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. RFC 3261: SIP: Session initiation protocol. http://www.ietf.org/rfc/rfc3261.txt?number=3261, June 2002.
11. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *In the Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99)*, pages 434–441, May 1999.
12. H. Wang, B. Raman, C. Chuah, R. Biswas, R. Gummadi, B. Hohlt, X. Hong, E. Kiciman, Z. Mao, J. Shih, L. Subramanian, B. Zhao, A. Joseph, and R. Katz. Iceberg: An internet-core network architecture for integrated communications. *IEEE Personal Communications Magazine*, 2000.
13. Web services: the cement for mobile, context-aware services. http://www.freeband.nl/projecten/wasp/ENindex.html (13-May-2004).