An Evaluation of the State of the Art in Context-aware Architectures

Pyrros Bratskas and Nearchos Paspallis and George A. Papadopoulos

University of Cyprus, Department of Computer Science {bratskas, nearchos, george}@cs.ucy.ac.cy

Abstract. Mobile computing is an innovative field gaining increasing attention as many new systems are designed towards that direction. Among these systems, many are desired to be context-aware, with the aim of optimizing and automating their offered services. Such systems provide components whose main feature is to manage the context information which is communicated between sensors, actuators and applications. In these systems the use of middleware is a solution to the need for detecting and adapting to the changing context. In mobile computing, factors such as scalability, support for distribution, self-adaptivity, support for mobility, modularity/plug-ability, etc are of particular interest. Many attempts have been documented in the literature concerning systems aiming to address some or all of these requirements and which are used for the implementation of context-aware systems. The scope of this paper is to study and present the current state of the art in context-aware system architectures. These are evaluated and compared, based on a set of characteristics such as support for distribution, privacy, mobility or fault tolerance. Finally, we document our current results and initial decisions concerning the design of a context management middleware system enabling the design and deployment of adaptive applications in mobile and ubiquitous computing environments.

1 Introduction

Mobile computing is an innovative field gaining increasing attention as many new systems are designed towards that direction. Among these systems, many are desired to be context-aware with the aim of optimizing and automating their offered services. Such systems provide components whose main functionality is to manage the context data which is communicated between sensors, actuators and applications. Furthermore, mobile computing environments include a huge spectrum of computation and communication devices that seamlessly aim to augment peoples' thoughts and activities with information, processing and analysis. Devices such as Personal Digital Assistants (PDAs) and mobile phones have gained a lot of popularity and are increasingly being networked. On the other hand, people use different software development platforms to create applications for this type of devices. These applica-

tions must be context-aware to meet the requirements of the highly distributed environment. These kind of context-aware systems adapt not only to changes in the environment, but also to user requirements. But even though the devices' capabilities are becoming more and more powerful, the design of context-aware applications is constrained not only by physical limitations, but also by the need to support a plethora of features such as distribution, scalability, modularity, mobility, privacy, fault tolerance etc. Indeed, mobile devices will continue to be battery-dependent and operate in an environment where more and more devices will be present and will need to communicate or share resources. Wide-area networking capabilities will continue to be based on communication with base stations and with fluctuations in bandwidth depending on physical location and on mobility.

In order to provide a fair quality of service to the users, applications must be context-aware and able to adapt to context changes, something that renders the development of context-aware applications an increasingly complex task. A solution to managing this complexity is the use of a middleware layer which is now becoming a common requirement. Many middleware solutions are proposed in the literature with the aim of addressing requirements which are normally addressed by middleware for distributed systems. They include support for sensing, gathering and managing context information in order to simplify application development through sharing this information among middleware components. However, as stated by (Henricksen, Indulska, McFadden, and Balasubramaniam 2005) many additional requirements are not met. In this paper, we evaluate and compare a set of popular approaches for implementing context-aware architectures. This is done with respect to a set of characteristics that were detected.

The rest of this paper is structured as follows. Section 2 briefly introduces an overview of the architecture of the examined systems. Then, Section 3 evaluates and compares the examined architectures based on a predefined set of desired features and characteristics. Following that evaluation, Section 4 describes the current state of our work, which aims at designing and implementing a context management system targeting adaptive applications in mobile and ubiquitous computing environments. Finally, Section 5 closes with our conclusions.

2 Context-aware Systems

Context-aware computing is an area which studies methods and tools for discovering, modeling and utilizing contextual information. Such information may include any information affecting the interaction of a user with a system, such as user location, time of day, nearby people and devices, user activity, light or noise conditions, etc. A more formal and widely used definition by (Dey 2000; Dey 2001) specifies context as "*any information that can be used to characterize the situation of an entity; an entity is a person, place, or object that is considered relevant to the interac-*

tion between a user and an application, including the user and application themselves".

Context can also be classified in more fine-grained categories: *physical*, *computing* and *user* context information types, as it is described by (Chen and Kotz 2000). The physical context type is related to environmental factors which can be usually evaluated by specialized hardware mechanisms. The light, noise, and temperature are examples of physical context data types. The computing context refers to the information which describes the resources available by the computing infrastructure. This includes information such as the network connectivity and its characteristics (e.g. bandwidth, latency, etc), nearby resources (such as printers, video projectors, etc), and details concerning its own memory availability, processor use, etc. Finally, the user context refers to the user's profile by focusing on the user needs, preferences, mood, etc. For instance, these can include information concerning the user's occupation (e.g. driving or studying) or the user's choice for preferring, say, to use a desktop computer rather than a PDA while at work.

Furthermore, as it is argued by (Satyanarayanan 2001), any system that aims to be minimally intrusive must be context aware, in the sense that it should be conscious of its user's state and environment. In other words, context-aware mobile systems are expected to utilize such information in order to adapt their behavior, based on a predefined set of adaptation rules. These rules are usually monitored by a system which dynamically adapts the system's operation based on the contextual information sensed.

In this section we present and review popular, middleware-based solutions which are used for enabling context-aware systems. This review is primarily aiming to evaluate a number of desired features for context-aware systems, as they are discussed in the next section.

2.1 The Context Toolkit

The Context Toolkit, introduced by (Dey, Salber, and Abowd 2001), assists software developers by providing them with a set of abstractions which enable them to build context-aware applications. These abstractions include: widgets, aggregators, interpreters, services and discoverers.

A widget is a component that is responsible for acquiring context information directly from a sensor. The aggregators can be thought of as meta-widgets, taking on all capabilities of widgets. They also provide the ability to aggregate context information of real world entities such as users or places and act as a gateway between applications and widgets. Interpreters transform low-level information into higher-level information that is more useful to applications. Services are used by context-aware applications to invoke actions using actuators, and discoverers can be used by applications to locate suitable widgets, interpreters, aggregators and services.

The toolkit is implemented as a set of Java APIs that represent its abstractions and use the HTTP protocol for communication and XML as the language model.

Components implemented in other languages can also interoperate through the use of Web standards. An extension to the Context Toolkit was proposed by (Newberger and Dey 2003) for providing user control of context-aware systems using an approach called end-user programming,

2.2 The Context Fusion Networks

The Context Fusion Networks (CFN) approach was proposed by (Chen and Kotz 2004). It allows context-aware applications to select distributed data sources and compose them with customized data-fusion operators into a directed acyclic information fusion graph. Such a graph represents how an application computes high-level understandings of its execution context from low-level sensory data. The CFN aims to address four challenges: flexibility, scalability, mobility and self-management.

CFN was implemented by a prototype named Solar, a flexible, scalable, mobility-aware, and self-managed system for heterogeneous and volatile ubiquitous computing environments. Solar consists of a set of functionally equivalent nodes, coded Planets, which peer together to form a service overlay using a distributed hash table (DHT) based P2P routing protocol such as Pastry. This approach is described by (Rowstron and Druschel 2001). A Planet has two kinds of message transports: normal TCP/IP based and DHT based services. Solar does not restrict the syntax of its events, as long as it follows a hierarchical attribute structure. Thus it is easy to incorporate XML events into the Solar system. Sensors and applications may connect to any Planet. Planets are execution environments for operators and they cooperatively provide several operator-management functionalities, such as naming and discovery, routing the sensory data through operators to applications, operator monitoring and recovery in face of host failures, and garbage collecting operators that are no longer in use. Each one of the Planet functionalities is considered a service. Thus, mobility service tracks clients (sensors or applications), which may detach from a Planet and reattach to another one.

2.3 The Context Fabric

The Context Fabric (Confab), a toolkit for facilitating the development of privacy-sensitive, ubiquitous computing applications was proposed by (Hong and Landay 2004). Confab focuses on privacy rather than on context sensing and processing. From a software architecture view, Confab provides a framework of privacy mechanisms that allow developers and end-users to support a spectrum of trust levels and privacy needs. Privacy and security are supported by the fact that personal information is captured, stored, and processed on the end-user's computer as much as possible.

Context information is modeled as Infospaces which can contain both context data and sources of the context data, such as for example sensors. Infospaces contain tuples that describe individual pieces of contextual data. These tuples are imple-

mented as data-centric XML documents that is, context tuples consist only of XML tags and XML attributes, with no text between tags. XPath is used as the query language for matching and retrieving XML tuples. Confab is implemented in Java and uses HTTP for network communication. It does not address traditional distributed system requirements such as mobility, scalability, component failures and deployment/configuration.

The core services include the Context Event Service which provides a universal event system for context events, the Context Query Service which provides a universal interface by which applications can synchronously check context state, the Automatic Path Creation Service which takes a directive in the Context Specification Language (CSL) and then processes it to the best of its abilities, and the Sensor Management Service which registers all local sensors and provides a uniform data format for sensors for the Context Event Service and the Context Query Service. The Context Specification Language is a simple XML-based language which provides a flexible way of specifying context needs.

2.4 Gaia

Roman et al (Roman et al 2002) have proposed Gaia which is designed to support the development and execution of portable applications for active spaces, which are programmable ubiquitous environments in which users interact with several devices and services simultaneously. Gaia manages the resources and services of an active space and has three components: the kernel, the application framework and the application. It offers five basic services: the *event manager* which distributes events in the active space and uses the CORBA event service as the default event factory, the *context service* through which applications query and register for context information (several components named context providers offer information about context), the *presence service* which detects and maintains information about software components and devices, the *space repositories* which store information about the entities in the space and lets applications browse and retrieve them based on specific attributes, and the *context file system* which is a context-aware file system and uses properties and environmental context information to simplify many of the tasks that are traditionally performed manually or require additional programming

Gaia uses XML to describe the resources of active spaces and LuaOrb, a high-level scripting language, to program and configure them. Corba services are widely used, however active spaces require extensions to handle fault tolerance and dynamic resource detection. Gaia supports direct communication similar to synchronous low-level operating system (OS) communication, and indirect communication similar to asynchronous low-level OS inter-process communication.

Gaia does not address scalability and privacy is not addressed by any of the basic services, but can potentially be provided by additional services. Heterogeneity, mobility and component configuration can all be supported by Gaia, but in limited forms.

2.5 Reconfigurable context-sensitive middleware

The *Reconfigurable Context-Sensitive Middleware* (RCSM) is a middleware proposed in (Yau et al 2002) and designed to provide two properties to applications: context-awareness and ad-hoc communication. This is done not in an independent way but in a way that allows RCSM to provide another property named context-sensitive ad-hoc communication. RCSM provides an object-based framework for supporting context-sensitive applications similar to middleware standards and prototypes such as CORBA, COM, and TAO for fixed networks.

Thus, RCSM provides application developers with a context-aware Interface Definition Language (CA-IDL) that can be used to specify context requirements, including the types of context that are relevant to the application, the actions to be triggered, and the timing of these actions. Ad hoc communication support is provided by a context-sensitive object request broker (R-ORB). This communicates at runtime with the skeletons produced by the compilation of the IDL interfaces, provides device and service discovery and use a symmetric communication model to allow ad hoc and application-transparent information exchange between a pair of remote objects.

The prototype described by Yau et al. does not satisfy the heterogeneity requirement, as it supports C++ applications only, for the Windows CE platform. However, the IDL compiler could potentially be modified to produce skeletons for a variety of platforms and communication protocols. In addition, the context discovery protocol is not flexible enough to support mobility or component failures, the RCSM authors do not attempt to address scalability, privacy, traceability, or control.

2.6 The PACE middleware

The PACE middleware consist on a set of components and tools that have been developed according to three design principles: First, the model of context information used in a context-aware system should be explicitly represented within the system. Second, the context-aware behavior of applications should be determined, at least in part, by external specifications that can be customized by users and evolved along with the context. Finally, the communication between application components and between the components and middleware services should not be tightly bound to the application logic. The components and tools that are part of the middleware include: a *context management system* which provides aggregation and storage of context information and performs query evaluation, a *preference management system* that provides decision-support for context-aware applications, a *programming toolkit* implemented in Java and using RMI for communication, that facilitates interaction between application components and the context and preference management systems, *tools* that assist with generating components and with developing and deploying context-aware systems, starting from context models specified in the context management system and finally a *messaging framework* for remote components to

communicate. The PACE middleware provides support for heterogeneity, mobility, traceability, control, deployment, configuration and decision-support. Requirements like scalable deployment, configuration and management of sensors have not been addressed.

Hardian (Hardian 2006) has enriched the middleware developed by (Henricksen et al 2005) by providing traceability and control to facilitate user understanding and feedback. This includes selectively exposing various components (context information, preferences, and adaptation rules and logic) to users. The new functionality can be viewed conceptually as an additional layer above the context and preference management components, providing logging and generation of explanations/feedback for users.

2.7 The MADAM context system

The main concept of the MADAM context system's architecture is the separation of concerns between context clients and context providers, as it is hinted by (Satyanarayanan 2001) and discussed by (Mikalsen et al 2006). In this respect, all nodes act as both context providers and context consumers, as part of a membership group which is formed using a loosely coupled protocol. Furthermore, while individual nodes are free to access context information from any possible provider it is nevertheless assumed that in most cases context sharing is limited to a local area only. In this respect, the locality refers to groups formed by nodes which can directly communicate with each other, e.g. over a wireless link by forming an ad-hoc WiFi or Bluetooth networks.

The MADAM context system architecture is built around a centralized context repository, where the context information is modeled and stored. Special entities are then used to populate the repository with context information (i.e. context sensors) and for registering and receiving notifications when certain context elements change (i.e. context listeners). Additionally, specially designed components (i.e. membership managers) are designed and implemented with the aim of enabling distribution of context information between different computing models.

This approach has the important advantage of assigning higher importance to local context and consequently enabling localized scalability. The first one refers to the fact that it is more likely that two neighboring nodes will share a common interest on the same context as opposed to nodes at different geographical locations. This is true for example in most pervasive computing applications where applications aim to utilize the infrastructure which is embedded in the surrounding environment. Second, localized scalability is achieved by preferring local sources (and respectively consumers) for sharing context information with. In this approach, the use of backbone links is avoided as most of the communication is carried out over local (i.e. direct) network links. The following paragraphs describe the basic ideas of this approach, along with the algorithms required to support it.

3 Evaluating and Comparing the State of the Art

In this section we compare and analyze the context-aware systems presented in the previous section in terms of their architecture. When designing a middleware system, many of the requirements of the classic distributed systems must also be taken into consideration: mobility, scalability, fault tolerance or/and heterogeneity. But additional requirements are also needed when designing context-aware systems.

Support for distribution. When dealing with context, the devices used to sense context most likely are not attached to the same computer running the application. The sensors may be physically distributed and cannot all be directly connected to a single machine. In addition, multiple applications may require use of that location information and these applications may run on multiple computing devices.

Modularity/Plug-ability. Modularity makes the system easy to extend through the introduction of new sensors, new devices, and new services. This feature is crucial due to the dynamic nature of context and in particular of user preferences. On the other hand, plug-ability concerns the feature of the system to support the ability to develop and/or easily install new context and sensor components like plug-ins, and eventually to combine them in complex hierarchies to support new context features, and offer new services as well.

Support for privacy/security. The use of distributed resources provides a huge potential for expanding the way that people communicate and share data, provide services to clients and process information to increase their efficiency. This broad access has also brought with it new security and privacy vulnerabilities.

Technology used for context modeling. Context modeling is about providing a high level abstraction of context information. Context information can be originated from a wide variety of sources, leading to extreme heterogeneity in terms of quality. In our analysis of the systems we focus on the technology they use e.g. XML as the interface description and encoding language or simple textual key-value pairs, to model the context information.

Technology used for communication. We refer to this requirement in our review to define which protocols are used for communication between components.

Support for mobility. All components (especially sensors and applications) can be mobile, and the communication protocols that underpin the system must therefore support appropriately flexible forms of routing. Context information may need to migrate with context-aware components.

Tolerance to failures. Sensors and other components are likely to fail in the ordinary for operation of a context-aware system. Disconnections may also occur. The system must continue operation, without requiring excessive resources to detect and handle failures.

Support for decision-making. This requirement refers to tools that help applications to select appropriate actions and adaptations based on the available context information.

The Context Toolkit defines a distributed architecture supporting context fusion and delivery with three notions of widget, aggregator and interpreter based on an object-oriented architecture. These are implemented as Java objects which communicate using a protocol based on HTTP and XML. However, there are a set of limitations of the context toolkit architecture. The main ones are the non-support for continuous context, unreliable data and context privacy, scalability or tolerance to failures and decision support.

The Context Fusion Networks was implemented as a prototype infrastructure named Solar. This infrastructure supports scalability and mobility and uses a P2P routing protocol for communication. Solar provides an XML-based composition language for application, identifying context-fusion operators and name queries that select sources. The language allows components to be connected using pull or push channels. However, Solar does not address privacy and decision support.

The Context Fabric has a decentralized architecture which primary concern is privacy. This is done with several customizable privacy mechanisms. Context fabric also comes with extensions for managing location privacy. Combined, these features allow application developers and end-users to support a spectrum of trust levels and privacy needs. While mobility, scalability, tolerance to failures and decision support are not addressed. Context fabric provides a framework and an extendable suite of mechanisms that application developers and end-users can use for managing privacy. From this point of view, there is a kind of modularity supported by the Context Fabric.

Gaia is a system built as a distributed middleware infrastructure that coordinates software entities and heterogeneous networked devices contained in a physical space. Gaia provides a framework to develop user-centric, resource-aware, multi-device, context-sensitive, and mobile applications. It supports mobility, but does not address scalability, privacy and decision support.

RCSM tries to provide a balance between awareness and transparency and support for ad hoc communications. RCSM allows addition or deletion of individual active object containers (ADCs) during runtime (to manage new or existing context-sensitive application objects) without affecting other runtime operations inside it. RCSM does not address scalability, privacy and decision support.

PACE Middleware offers support for mobility through the use of Elvin by (Segall, Arnold, Boot, Henderson, and Phelps 2000) a content-based message routing scheme, which facilitates component mobility. Privacy is addressed by providing access control to sensitive information. PACE, unlike the others solutions presented above, provides also decision support. On the other hand there is no support for scalability and tolerance to failures is not fully supported.

The evaluation results are summarized in Table 1. Although there are no definite answers as per the optimal solution to specific requirements (e.g. we cannot argue for sure whether XML-based modeling is preferable to Object-based modeling), however it is possible to argue that some approaches are more *complete* as per the predefined requirements as compared to others. For example, the Context Fabric approach

appears to satisfy only a few of the requirements that we have predefined, which is natural observing that the main direction of its designers was to achieve privacy and security related characteristics. On the other hand, approaches such as the PACE and the MADAM projects appear to be more promising with respect to the prospects they offer for inspiring a new approach for which these requirements are important.

Table 1: Comparing the studied context management systems

Requirement	Context Toolkit	CFN	Context Fabric	Gaia	RCSM	PACE	MADAM
Support for distribution	Yes	Yes	No	Yes	Yes	Yes	Yes
Modular-ity/Plug-ability	No	No	No	No	No	No	Yes
Privacy /Security	No	No	Yes	No	No	Yes	Low
Mobility	Low	Yes	No	Low	No	Yes	Yes
Tolerance to failures	No	Yes	No	Low	No	Low	Yes
Scalability	No	Yes	No	No	No	No	Low
Decision sup-port	No	No	No	No	No	Yes	No
Technology for Context modeling	XML	XML based	CSL (XML based)	XML, Lua ORB	CA-IDL	CML (XML based)	Object-based
Technology for Communication	HTTP	TCP/ IP	HTTP	CORB A	R-ORB	HTTP	Java RMI

4 Towards a new Context Management System

The review of context architectures in this paper was undertaken as part of our effort to define the architecture of the MUSIC middleware system. This system aims at providing software engineers with appropriate methods and tools for the efficient development of context-aware, adaptive applications. Because MUSIC primarily targets at applications that are deployed in mobile and ubiquitous computing environments, we started quite early by trying to detect the most important requirements which are critical for these environments.

For example, the fact that some of the involved applications and devices are expected to be embedded in the infrastructure, as per the ubiquitous computing paradigm, imposes the requirement for efficient, small-footprint context management systems. Additionally, as other devices might be expected to require more function-

alities, and consequently more complex systems, the designed architecture should be as modular as possible. For instance, target devices include smart-phones, PDAs, and laptops. The first are small and require efficient, small-footprint applications. The latter are typically very powerful devices and thus should not limit their functionalities simply for efficiency reasons. This validates our decision to set scalability and plug-ability as a high priority in our approach.

For the future, we have started working on three primary directions. The first one is the design of an appropriate context model, which can be expressed by programmatic Objects (at runtime), by XML artifacts (during communication and while stored on disk) and by UML artifacts (during design time). Furthermore, we are working on studying different approaches for distribution of context information on top of several existing technologies such as WiFi, Bluetooth, JXTA (i.e. peer-to-peer), Jini, UPnP, etc. Finally, we are working on defining special languages for enabling context querying and context accessing. These languages are significant as we are aiming to provide a complete methodology for enabling developers to design the context-awareness aspect of their applications in an automated way, as per the MDA approach.

5 Conclusions

The main goal of this paper was to research and evaluate existing context-management architectures, as they were proposed and presented in the literature with the aim of assisting and directing the design and implementation of a novel context management system. The newly designed system aims at providing support for designing and engineering adaptive applications targeting mobile and ubiquitous computing environments. For this reason, we extracted a set of important features that we wanted to compare these projects on, and we extracted results that are currently used while we are designing the MUSIC context system.

Although a number of existing approaches appear to be quite suitable and efficient, none of them appears to be a perfect match for the detected requirements. Nevertheless, we have scheduled a design and implementation process which leverages on the knowledge extracted by these systems so that we avoid solving problems that have already been tackled. Finally, we have presented our current work and described our plans for future work, in the frame of the MUSIC project. These plans include research in the areas of modeling, managing and distributing context information, which was also partly faced by the studied approaches.

References

Chen, G., Kotz, D. (2000) A Survey of Context-Aware Mobile Computing Research, Technical Report: TR2000-381 Dartmouth College, Hanover, NH, USA.

Chen, G., Li, M., Kotz, D. (2004) Design and implementation of a large-scale context fusion network. 1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous), IEEE Computer Society, pp. 246–255.

Dey, A. (2000) Providing Architectural Support for Building Context-Aware Applications, PhD Thesis, College of Computing, Georgia Institute of Technology, pp. 170.

Dey, A. K., Salber, D., and Abowd, G. D. (2001) A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, Human-Computer Interaction, 16, pp, 1-67.

Dey, A. (2001) Understanding and Using Context, Personal Ubiquitous Computing, 5(1), pp. 4-7.

Hardian, B. (2006) Middleware support for transparency and user control in context-aware systems. 3rd international Middleware Doctoral Symposium, Melbourne, Australia, Nov. 27 – Dec. 1, 2006. MDS '06, vol. 185. ACM Press, New York, NY, 4.

Henricksen, K., Indulska, J., McFadden, T., Balasubramaniam, S. (2005) Middleware for Distributed Context-Aware Systems, International Symposium on Distributed Objects and Applications (DOA), Ayia Napa, Cyprus, Oct 31st-Nov 4th, 2005, pp. 846-863.

Hong, J.I., Landay, J.A. (2004) An Architecture for Privacy-sensitive Ubiquitous Computing, 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys), Boston.

Mikalsen, M., Paspallis, N., Floch, J., Stav, E., Papadopoulos G. A., Ruiz, P. A. (2006) Putting Context in Context: The Role and Design of Context Management in a Mobility and Adaptation Enabling Middleware, International Workshop on Managing Context Information and Semantics in Mobile Environments (MCISME'06), Nara, Japan, May 9-12, 2006, IEEE Computer Society Press, pp. 76-83.

Newberger, A., Dey, A. (2003) Designer Support for Context Monitoring and Control, Intel Research Berkeley.

Paspallis N., Papadopoulos, G. A. (2006) An Approach for Developing Adaptive, Mobile Applications with Separation of Concerns, 30th International Computer Software and Applications Conference (COMPSAC 2006), Chicago, IL, USA, Sept. 17-21, 2006, IEEE Computer Society Press, pp. 299-306.

Rowstron, A., Druschel., P. (2001) Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, International Middleware Conference, Heidelberg, Germany, pp. 329–350.

Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H., Nahrstedt, K. (2002) Gaia: A middleware infrastructure for active spaces. IEEE Pervasive Computing, Special Issue on Wearable Computing 1, pp. 74–83.

Satyanarayanan, M. (2001) Pervasive Computing: Vision and Challenges, IEEE Personal Communications Magazine, pp. 10-17.

Segall, B., Arnold, D., Boot, J., Henderson, M., Phelps, T. (2000) Content based routing with Elvin4, AUUG2K Conference, Canberra.

Yau, S. S., Karim, F., Wang, Y., Wang, B., and Gupta, S. (2002) Reconfigurable Context-Sensitive Middleware for Pervasive Computing, IEEE Pervasive Computing, pp. 33-40.