Next Generation Mobile Networks and Ubiquitous Computing

Samuel Pierre
École Polytechnique de Montréal, Canada

Information Science REFERENCE

INFORMATION SCIENCE REFERENCE
Hershey · New York

Director of Editorial Content: Kristin Klinger
Director of Book Publications: Julia Mosemann
Acquisitions Editor: Lindsay Johnston
Development Editor: Christine Bufton, Juli Mosemann
Publishing Assistant: Milan Vracarich
Typesetter: Travis Gundrum, Deanna Jo Zombro
Production Editor: Jamie Snavely
Cover Design: Lisa Tosheff

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Next generation mobile networks and ubiquitous computing / Samuel Pierre, editor.
p. cm.
Includes bibliographical references and index. Summary: "This book provides a comprehensive and unified view of the latest and most innovative research findings on the many existing interactions between mobile networking, wireless communications, and ubiquitous computing"--Provided by publisher. ISBN 978-1-60566-250-3 (hardcover) -- ISBN 978-1-60566-251-0 (ebook)
1. Ad hoc networks (Computer networks) 2. Ubiquitous computing. I. Pierre, Samuel.

TK5105.77.N49 2010
621.382--dc22
2010010158

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 12
Middleware Technologies for Ubiquitous Computing

Noha Ibrahim
Université de Lyon, France

Frédéric Le Mouël
Université de Lyon, France

Stéphane Frénot
Université de Lyon, France

ABSTRACT

Middleware handles many important functionalities for ubiquitous computing. The authors distinguish various middleware technologies providing key elements for all applications' requirements such as discovery, adaptation/composition, context management, and management of ubiquitous applications. In this chapter, they propose a classification for some of the most employed ubiquitous middleware. The classification was established upon the challenges raised by ubiquitous computing – effective use of smart spaces, invisibility, and localized scalability - and upon how the various ubiquitous middleware respond to them in terms of interoperability, discoverability, location transparency, adaptability, context awareness, scalability, security, and autonomous management. This classification shows that if many middleware are mature enough and offer specific functionalities respecting the properties of ubiquity, a real lack is noticed in having an interoperable, autonomous and scalable middleware for the execution of ubiquitous applications. The development of the service-oriented paradigm, the semantics, the Web middleware, and the ambient intelligence shows the new trend the middleware research field is engaged in.

INTRODUCTION

Middleware are enabling technologies for the development, execution and interaction of applications. These software layers are standing between the operating systems and applications. They have evolved from simple beginnings — hiding network details from applications — into sophisticated systems that handle many important functionalities for distributed applications — providing support for distribution, heterogeneity and mobility. The evolution of middleware has been influenced by numerous developments and standards efforts. Various middleware paradigms were defined. We

DOI: 10.4018/978-1-60566-250-3.ch012

cite the Remote Procedure Call middleware (RPC), the Message Oriented Middleware middleware (MOM), the Object Request Broker middleware (ORB), and the Service-Oriented Architecture middleware (SOA).

The term middleware first appeared in the late 1980s to describe network connection management software, but did not come into widespread use until the mid 1990s, when network technology had achieved sufficient penetration and visibility. By that time, middleware had evolved into a much richer set of paradigms and services, offered to build distributed applications more easily and in a more manageable way. The term was associated mainly with relational databases for many practitioners in the business world through the early 1990s. By the mid-1990s this was no longer the case. Concepts similar to today's middleware previously went under the names of network operating systems, distributed operating systems and distributed computing environments. Systems such as Apollo's Network Computing Architecture (NCA), Sun's RPC standard, and the Open Software Foundation's Distributed Computing Environment (DCE) are all examples of successful RPC-oriented middleware that has been used for significant production applications. At the same time these systems were being produced in the 1980s and early 1990s, significant research was occurring in the area of distributed objects. Distributed objects represented the confluence of two key areas of information technology: distributed systems and object-oriented design and programming. As a result of research on these and other distributed object systems, combined with an evolution of the Transaction Processing monitor (TP monitor) into the concept of an "object monitor," Common Object Request Broker Architecture (CORBA) was created. The CORBA system is built on top of ORB, which encompasses the entire communication infrastructure necessary to identify and locate objects, handle connection management, and deliver data. At the same time, a specific class of middleware, MOM, that operates on the principles of message passing or message queuing appeared. The MOM middleware unlike RPC and object-orientation is an asynchronous form of communication, i.e. the sender does not block waiting for the recipient to participate in the exchange. While the evolution of distributed objects in the 1990s was marked by significant efforts to establish standards such as CORBA, the evolution of messaging oriented middleware was practically devoid of standards efforts.

In the late 1990s, the growing popularity of Java, the explosive growth of the World Wide Web (WWW), and lessons learned from CORBA and messaging were all combined to form the Java 2 Enterprise Edition (J2EE), a comprehensive component middleware platform. J2EE provides support for numerous application types, including distributed objects, components, web-based applications, and messaging systems. Throughout the development of CORBA, J2EE, and proprietary messaging systems, Microsoft was busy developing its Distributed Component Object Model (DCOM).

In 1999 and 2000, the Web's influence on middleware started to become readily apparent with the publication of the initial version of Simple Object Access Protocol (SOAP). Initially dubbed SOAP was the result of trying to create a system-agnostic protocol that could be used over the Web and yet still interface easily to non-SOAP middleware, including CORBA, DCOM, J2EE, and messaging middleware systems. Given the 1990s "middleware wars" between J2EE, CORBA, and DCOM, and between RPC and messaging, the unified support for SOAP was indeed groundbreaking. At the time, it seemed that SOAP, Web Services and Service-Oriented Computing in general might finally provide the basis for broad industry agreement on middleware standards.

BACKGROUND

If middleware were designed to help manage the complexity and heterogeneity inherent in distributed systems, one can imagine the new role middleware has to play in order to respect the evolution from distributed and mobile computing to ubiquitous one. Ubiquitous computing is all about everywhere and anytime computing. It spreads intelligence and connectivity to more or less everything. So conceptually, machines, clothing, tools, appliances, homes and even the human body will be embedded with chips to connect to an infinite network of other devices. This creates an environment where the connectivity of devices is embedded in such a way that it is unobtrusive and always available.

"*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it*" - so began Mark Weiser's (1991) paper (p.66) that described his vision of ubiquitous computing. The essence of that vision was the creation of environments saturated with computing and wireless communications capability, yet gracefully integrated with human users. If many of these features were unavailable in 1991, many key building blocks needed for this vision are now viable commercial technologies. Realizing the above vision of ubiquitous computing requires dealing with a number of issues, mainly due to the environment's heterogeneity, dynamics, and constraints. Middleware technologies homogenize the diversity of technologies in ubiquitous computing while providing a support to the development of ubiquitous computing software applications.

Ubiquitous computing raises many constraints that have an impact on middleware technologies. Mobile elements are more likely to be everywhere at the same time. Their ubiquity makes it difficult to consider their availability. It leads them to many different places and situations, much more complex than in a static position. Mobile elements are characterized by a variable connectivity. Indoors and outdoors may offer both reliable, high-bandwidth wireless connectivity, or low-bandwidth connectivity with gaps in coverage. Ubiquitous environments are populated with heterogeneous devices and providers, making interaction between these entities more difficult. While hardware devices will undoubtedly improve over time, the problem of power consumption will not diminish. These hazardous mobility, variable connectivity, heterogeneity, hardware constraints, and security and privacy problems affect the software systems and so the middleware, the glue for all software systems.

These constraints define challenges to be taken into account. We distinguish three challenges: effective use of smart spaces, invisibility and localized scalability.

By embedding computing infrastructure to almost everything and everywhere, smart spaces brings together the world of humans and the world of computing in a new unexplored way. If a pervasive computing environment continuously meets user expectations and rarely presents him with surprises, it allows him to interact almost at a subconscious level while leaving him the choice to interact with the system whenever he wants. Future pervasive computing environments will likely face a proliferation of users, applications, networked devices, and their interactions on a scale never experienced before. As environmental smartness grows, so will the number of devices connected to the environment and the intensity of human machine interactions. Scalability is thus a critical problem in pervasive computing. The density of interactions has to fall off as one moves away.

These challenges have already been a subject of researches for numerous fields such as networking and wireless communications, but also for the applications level such as pervasive home and domotic applications. We argue that middleware technologies need also to cope with these new environments, in order to support the development done in the networks and applications layers.

Requirements and main characteristics of middleware for ubiquitous computing were widely studied (Niemela, 2004, Mascolo, 2005, Salminen, 2005). These studies try to define basic requirements for ubiquitous middleware. We will focus on the most relevant characteristics of ubiquitous computing middleware. The following eight fundamental requirements can be identified: interoperability, discoverability, location transparency, adaptability, context awareness, scalability, security, and autonomous management.

Ubiquitous computing environments, quoting Mark Weiser's definition, consist of various kinds of computational devices, networks and collaborating software and hardware entities. Due to the large number of heterogeneous and cooperating parties, interoperability is required at all levels of ubiquitous computing. Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged.

One of the fundamental challenges of distributed and highly dynamic environments is how the applications can discover the surrounding entities and, conversely, how the applications can be discovered by the other entities in the system. Discoverability is thus a major issue for ubiquity.

In a ubiquitous system, the execution environment of applications can be logically considered a single container including all applications, other components, and resources. Moreover, the idea in distributed environments is that the resources can be accessed without any knowledge of where the resources or the user are physically located. Whether a file is located in the mobile terminal's local file system or in a network server, the user accesses it in the same way. This is called location transparency; the locations of the system components are transparent to the other components, the programmer, and the user.

Changes in applications' and users' requirements or changes within the network, may require the presence of adaptation mechanisms within the middleware. Adaptability is the ability of a software entity to adapt to the changing environment. Moreover, adaptation is necessary when a significant mismatch occurs between the supply and demand of a resource. Adaptability is regarded as one of the most important functionalities in ubiquitous computing. For example, some resources or services in the environment may be vital to the application. As the application's execution environment changes due to the user's mobility, the vital resources need to be substituted by corresponding resources in the new environment in order to ensure continuous operation. The requirement for adaptation is present on many different layers of a computing system.

Ubiquitous middleware need to be context aware in terms of devices coming and leaving, functionalities offered and retrieved, quality of service changing, etc. They need to be aware of all these changes, in order to offer the best functionalities to applications regardless the context around. When considering context-aware systems in general, some common functionalities can be identified that are present in almost every system: context sensing and processing, context information representation, and the applications that utilize the context information. In general, the context information can be divided into low- and high-level context information. Low-level context information can be collected using sensors in the system. Low-level context information sources can be combined or processed further to higher level context information.

Scalability is the ability of the system to accommodate a higher load at some time in the future. A system's load can be measured using many different parameters, such as the maximum number of concurrent users, the number of transactions executed in a time unit, the number of services available in the ubiquitous environment, etc. As mentioned above, scalability is an important challenge for ubiquitous systems.

Security mechanisms, such as authentication, authorization, and accounting (AAA) functions may be an important part of the middleware in

order to intelligently control access to computer and network resources, enforcing policies, auditing network/user usage, etc. Another important aspect concerns privacy and trust in ubiquitous environments. In presence of unknown devices, middleware need to respect privacy of users, and provide trust mechanisms adapted to the ever changing nature of the environment.

Autonomous Management concerns the ability for a ubiquitous middleware to control and manage its resources, functions, security and performance, in the face of failures and changes, with little or no human intervention. The complexity of future ubiquitous computing environments will be such that it will be impossible for human administrators to perform their traditional functions of configuration management, performability management, and security management. Instead, one must resort to automate most of these management functions, allowing humans to concentrate on the definition and supervision of high-level management policies, while the middleware itself takes care of the translation of these high-level policies into automated control structures. The challenge is therefore to move from classical middleware support for configuration, performability and security management to support for self-configuration, self-tuning, self-healing and self-protecting capabilities.

CLASSIFICATION OF THE UBIQUITOUS MIDDLEWARE

Several ubiquitous middleware architectures and infrastructures have been introduced in the academic and industrial world. The current middleware treat ubiquity from slightly different perspectives. We distinguish various middleware technologies, ranging from partially-integrated middleware to fully-integrated middleware. We mean by fully-integrated middleware, middleware providing key elements for all applications' requirements such as discovery, adaptation/composition, context management, and management of ubiquitous applications. In this category we cite ubiquitous middleware systems such as Aura (Garlan, 2002), Gaia (Chetan, 2005), Oxygen (Rudolph, 2001), Pcom (Becker, 2004), and One.world (Grimm, 2004). Partially-integrated middleware range from platforms that were specially realized to handle one or two ubiquitous requirements, such as the application discovery in Jini (Kumaran, 2002) and UPnP (UpnP, 2006), to platforms that are being extended to ubiquity for the application management such as OSGi (OSGi Alliance, 2005) and.Net Framework (Chappell, 2006, Prengel, 2006).

We survey the current state-of-the-art architectures from the viewpoint of the core requirements identified above. In this survey, we will highlight the most known and used fully and partially-integrated middleware. We will not deal with the platforms that are being extended to ubiquity as these extensions are still in a preliminary state. Later on, a classification will focus on the strength and weakness of each of the ubiquitous middleware, based on the identified requirements.

Aura (Garlan, 2002) provides user with an invisible halo of computing and information services that persists regardless of location. A personal Aura acts as a proxy for the mobile user it represents. Aura aim is to allow users to execute their tasks regardless their location. It allows users to dynamically realize daily tasks modelled as abstract software applications, in a transparent way, without manually dealing with the configuration and reconfiguration issues of these applications. Aura deals more with adaptation, replacement of services, the dynamic configuration and reconfiguration of user tasks. Project Aura provides several pervasive applications adapted to both homes and offices.

Gaia (Chetan, 2005) is a services-based middleware that integrates resources of various devices. It manages several functions such as forming and maintaining device collections, sharing resources among devices and enables

seamless service interactions. It also provides an application framework to develop applications for the device collection. The application framework decomposes the application into smaller components that can run on different devices in this collection. The notion of ad-hoc pervasive computing in Gaia is a cluster of personal devices that can communicate and share resources among each other. The cluster is referred to as a personal active space. The user can program this cluster through a common interface. Mobile Gaia role is to provide services that discover devices that form the personal space, maintain the composition of the cluster, share resources among devices in the cluster and facilitate communication. Similarly to Aura, Gaia focuses on the dynamic aspect of ubiquitous environments and provides the support for dynamically mapping applications to available ressources of a specific active space.

Oxygen (Rudolph, 2001) vision is to bring an abundance of computation and communication within easy reach of humans through natural perceptual interfaces of speech and vision. Computation blends into peoples' lives enabling them to easily do tasks they want to do, collaborate, access knowledge, automate routine tasks and their environment. In other words, it enables a pervasive, human centric computing. The approach focuses on four technological areas: embedded computational devices, handheld devices, networks, and also on adaptive software. Perception is a central issue, however the focus is mainly on vision and speech aiming to replace explicit traditional input mechanisms with conversational and gesture input.

One.world (Grimm, 2004) is a system architecture for ubiquitous computing. It provides an integrated, comprehensive framework for building pervasive applications. The One.world architecture builds on four foundation services. First, a virtual machine provides a uniform execution environment across all devices and supports the ad hoc composition between applications and devices. Second, tuples define a common type system for all applications and simplify the sharing of data. Third, events are used for all communications and make change explicit to applications. Applications are composed from components that exchange events through imported and exported event handlers. Events make change explicit to applications, with the goal that applications adapt to change instead of forcing users to manually reconfigure their devices and applications. Finally, environments host applications, store persistent data, and—through nesting—facilitate the composition of applications and services.

Pcom (Becker, 2004), a Component system for ubiquitous computing is a light-weight component system that offers application programmers a high-level programming abstraction which captures the dependencies between components using contracts. Pcom allows the specification of distributed applications that are made up of components with explicit dependencies modeled using contracts. Pcom relies on a communication middleware, Base. Base is a flexible middleware for Pervasive computing environments. It provides adaptation support on the communication level by dynamically selecting or reselecting communication protocol stacks, even for currently running interaction. Base is written in Java using the Java 2 Micro Edition with the Connected Limited Device Configuration. It assists application programmers by providing mechanisms for device discovery and service registration that can be used to locate and access local as well as remote device capabilities and services. It also provides a simple signaling mechanism to determine the availability of these devices and services.

Jini (Kumaran, 2002) is a Java-based architecture for spontaneous networking. Participants in a Jini community require no previously knowledge of each other, and can take full advantages of the dynamic class loading and type-checking of the Java language, which requires a Java virtual machine (JVM) for all participants. A Jini community is established around one or more LookupServices, which organize the services deployed in the community and respond to requests from clients. The

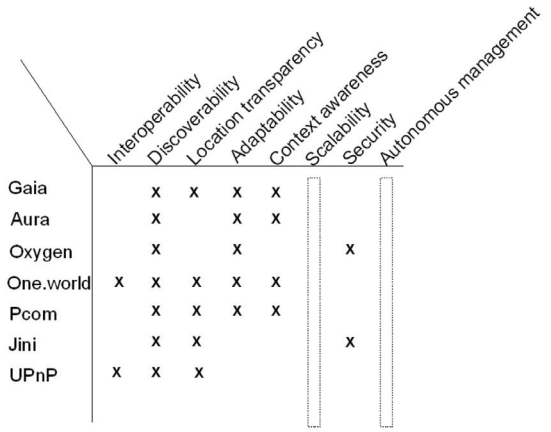
Lookup service is itself a Jini service, acting as a bootstrapping service. References to these Lookup services are obtained either by unicast or multicast discovery protocols defined by Jini. The main idea of Jini for supporting "spontaneous networking" is achieved by a leasing principle, which means that services are leased into the community. When a service provider registers a service in the Lookup service it obtains a lease, which must be renewed before it expires, otherwise the Lookup service automatically de-register the service. Clients can register for changes in the Jini community, such as new, discarded, or changed services, using remote event registrations. By the same principle clients and service providers can register for events of new or discarded Lookup services. Event-registrations are leased in the community, so automatic cleanup can be initiated for non-responding clients. These are the real benefits of Jini, enabling opportunity to create a self maintaining ubiquitous computing.

UPnP (UPnP, 2006) technology defines an architecture for ubiquitous peer-to-peer network connectivity of intelligent appliances, wireless devices, and PCs of all form factors. It is designed to bring easy-to-use, flexible, standards-based connectivity to ad-hoc or unmanaged networks whether in the home, in a small business, public spaces, or attached to the Internet. UPnP technology provides a distributed, open networking architecture that leverages TCP/IP and the Web technologies to enable seamless proximity networking in addition to control and data transfer among networked devices. It is designed to support zero-configuration, "invisible" networking, and automatic discovery for a breadth of device categories from a wide range of vendors. A device can dynamically join a network, obtain an IP address, convey its capabilities, and learn about the presence and capabilities of other devices. A device can leave a network smoothly and automatically without leaving any unwanted state behind.

We propose a classification of the previously mentioned ubiquitous middleware. The classification was established upon the challenges raised by ubiquitous computing and upon how the various ubiquitous middleware respond to them.

Figure 1, classifies the existent ubiquitous middleware defined above using the requirements of ubiquitous middleware. For each middleware technology, we focused on the requirements it respects and the ones it does not fulfill. If some requirements are relatively well fulfilled by nowadays systems, such as discoverability, context awareness and adaptability, others are far from being fulfilled or even dealt with such as interoperability, security, and especially scalability and autonomous management.

Figure 1. Classification of ubiquitous middleware



FUTURE TRENDS

Ubiquitous middleware offer more and more functionalities to applications evolving in ubiquitous environments and that by respecting some defined properties. Several domains and technologies have emerged as the new trends for ubiquitous middleware. These new trends tend to deal with the requirements left behind by the actual ubiquitous middleware. The semantic web (McIlraith, 2001), the ambient intelligence (Shadbolt, 2003) and the autonomic computing (Crowley, 2007) are

the most spread researches domain for ubiquitous computing.

Semantic modeling tries to solve the interoperability problem. The semantic web is an evolving extension of the World Wide Web in which web content can be expressed not only in natural language, but also in a format that can be read and used by automated tools, thus permitting people and machines to find, share and integrate information more easily. Today's Web was designed primarily for human interpretation and use. Nevertheless, we are seeing increased automation of Web service interoperation primarily in B2B and e-commerce applications. A fundamental component of the semantic web will be the markup of web services to make them computer-interoperable, user-apparent, and agent-ready.

Ambient intelligence refers to electronic environments that are sensitive and responsive to the presence of people. The ambient intelligence paradigm builds upon ubiquitous computing and human-centric computer interaction design and is characterized by systems and technologies that are embedded, context aware, personalized, adaptive and anticipatory. Ambient intelligence introduces proactivity to the world of ubiquitous computing. A proactive middleware needs to interfere in humans everyday, while respecting the invisibility imposed by ubiquitous computing. A balance needs to be found between invisibility and proactivity. Ambient intelligence builds on three recent key technologies: ubiquitous computing, ubiquitous communication and intelligent user interfaces.

Autonomic computing is an initiative started by IBM in 2001. Its ultimate aim is to create computer systems capable of self-management, to overcome the rapidly growing complexity of computing systems management, and to reduce the barrier that complexity poses to further growth. In a self-managing autonomic system, the human operator takes a new role, he does not control the system directly. Instead, he defines general policies and rules that serve as an input for the self-management process. For this process, IBM proposed future computer system not only to adapt to changing environments at runtime but further to be self-configuring, self-healing, self-optimizing and self-protecting. Autonomic computing is a promising opening research field for ubiquitous computing as autonomy can be appreciated by users desiring to use functionalities offered by these environments.

CONCLUSION

Ubiquitous middleware are becoming the nowadays trend in the development of ubiquity in computer science fields. Ubiquitous applications rely upon this layer, to profit from the diverse functionalities it has to offer. Ubiquitous environments brought more constraints and challenges to mobile environments. The main constraints come from, the environment's heterogeneity and dynamics, and the variable connectivity of the devices coming and leaving. The main challenges are in maintaining the computing smartness, scalability, invisibility and proactivity for the users in these environments. The functionalities offered by middleware need to cope with these challenging nature of environments. We sorted the middleware in two groups. The fully-integrated ones, provide functionalities such as discovery, adaptation/composition, and context management. The partially-integrated ones, provide one or two of these functionalities, as they were specifically developed for a specific purpose. We classified these middleware, by analysing if they are interoperable, discoverable, adaptable, context aware, scalable, secure and autonomous. If many of these middleware are mature enough and offer specific functionalities respecting the properties of ubiquity, a real lack is noticed in having an interoperable, autonomous and scalable middleware for the execution of ubiquitous applications. The development of the service-oriented paradigm, the semantics and the Web middleware shows

the new trend the middleware research field is engaged in. At the other hand the intersection of this research field with artificial intelligence and autonomic computing leads to the development of the ambient intelligence, the future evolution of ubiquitous computing.

REFERENCES

Becker, C., Handte, M., Schiele, G., & Rothermel, K. (2004). PCOM – A Component System for Pervasive Computing. *2nd IEEE Annual Conference on Pervasive Computing and Communications (PERCOM'04)*, Washington, DC, USA.

Chappell, D. (2006). Introducing the.Net Framework 3.0. Whitepaper "Introducing the.Net Framework 3.0".

Chetan, S., Al-Muhtadi, J., Campbell, R., & Mickunas, M. D. (2005). Mobile Gaia: A Middleware for Ad-hoc Pervasive Computing. *IEEE Consumer Communications & Networking Conference (CCNC 2005)*, Las Vegas, USA.

Crowley, J. L., Hall, D., & Emonet, R. (2007). Autonomic Computer Vision Systems. *International Conference on Computer Vision Systems (ICVS'07)*, Bielefeld, Germany.

Garlan, D., Siewiorek, D., Smailagic, A., & Steenkiste, P. (2002). Project Aura: Towards distraction-free pervasive computing. *IEEE Pervasive Computing / IEEE Computer Society [and] IEEE Communications Society*, *21*(2), 22–31. doi:10.1109/MPRV.2002.1012334

Grimm, R. (2004). One.World: Experiences with a Pervasive Computing Architecture. *IEEE Pervasive Computing / IEEE Computer Society [and] IEEE Communications Society*, *3*(3), 22–30. doi:10.1109/MPRV.2004.1321024

Kumaran, S., I. (2002). JINI Technology an Overview. *Prentice Hall PTR*.

Mascolo, C., Hailes, S., Lymberopoulos, L., Picco, G.P., Costa, P., Blair, G., Okanda, P., Sivaharan, T., Fritsche, W., Karl, M., Rnai, M.A., Fodor, K., & Boulis, A. (2005). *Survey of middleware for networked embedded systems*. (Technical Report D.5.1).

McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic Web Services. *IEEE Intelligent Systems*, *16*(1), 46–53. doi:10.1109/5254.920599

Niemela, E., & Latvakoski, J. (2004). Survey of requirements and solutions for ubiquitous software. *3rd international conference on Mobile and ubiquitous multimedia (MUM '04)* (pp. 71-78). New York.

OSGi Alliance. (2007). OSGi Service Platform. *Core Specification Release 4.1*. Retrieved from http://www.osgi.org/Specifications/HomePage

Prengel, F. (2006). *Net Micro Framework: Programming Small Connected Devices with*. Net, Core Specification. [White paper]

Rudolph, L. (2001). Project Oxygen: Pervasive, Human-Centric Computing - An Initial Experience. *Advanced Information Systems Engineering, 13th International Conference (CaiSE2001)* (LNCS 2068, pp. 765-780) Interlaken, Switzerland.

Salminen, T. (2005). *Lightweight middleware architecture for mobile phones*. Diploma Thesis, University of Oulu, Department of Electrical and Information Engineering.

Satyanarayanan, M. (2001). Pervasive Computing: Vision and Challenges. *IEEE Personal Communication, 8*(4), 10-17.

Shadbolt, N. (2003). Ambient Intelligence. *IEEE Intelligent Systems*, *18*(1), 2–3. doi:10.1109/MIS.2003.1200718

UPnP forum (2006). *UPnP Technology – the simple, seamless home network* [White paper].

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, *265*(3), 66–75. doi:10.1038/scientificamerican0991-94

KEY TERMS AND DEFINITIONS

B2B: (Business-to-business) is a term commonly used to describe electronic commerce transactions between businesses, as opposed to those between businesses and other groups, such as business and individual consumers (B2C) or business and government (B2G).

CORBA: (Common Object Request Broker Architecture) is an open infrastructure for distributed systems that is standardized by the Object Management Group (OMG). CORBA specifies a system which provides interoperability between objects in a heterogeneous distributed environment. Furthermore, it provides transparency from the network, different operating systems, and different programming languages in a distributed computing environment.

DCOM: (Distributed Component Object Model) is a proprietary Microsoft technology for communication among software components distributed across networked computers. DCOM, extends Microsoft's COM, a platform for software componentry introduced by Microsoft in 1993.

MOM: (Message Oriented Middleware) is a client/server infrastructure that increases the interoperability, portability, and flexibility of an application by allowing the application to be distributed over multiple heterogeneous platforms. It resides in both portions of client/server architecture and typically supports asynchronous calls between the client and server applications. Message queues provide temporary storage when the destination program is busy or not connected.

ORB: (Object Request Broker) is a piece of middleware software that allows programmers to make program calls from one computer to another via a network. ORB's handle the transformation of in-process data structures to and from the byte sequence, which is transmitted over the network. This is called marshalling or serialization.

RPC: (Remote Procedure Call) is a technology that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer or a shared network) without the programmer explicitly coding the details for this remote interaction.

SOA: (Service-Oriented Architecture) is a computer systems architecture style for creating and using business processes, packages as services, throughout their life cycle. Main characteristics of the service-oriented architecture are its support for the deployment and the interaction of loosely coupled software systems, which evolve in a dynamic and open environment and can be composed with other services.

SOAP: (Simple Object Access Protocol) is a protocol for exchanging XML-based messages over computer networks, normally using HTTP/HTTPS. SOAP forms the foundation layer of the web services protocol stack providing a basic messaging framework upon which abstract layers can be built.

TP Monitor: (Transaction Processing monitor) is a program that monitors a transaction as it passes from one stage in a process to another. Its purpose is to ensure that the transaction processes completely, or, if an error occurs, to take appropriate actions.

Web Service: is defined by the W3C as a software system designed to support interoperable Machine to Machine interaction over a network. Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.