ESPOO 2004 VTT RESEARCH NOTES 2238

Päivi Kallio, Eila Niemelä & Juhani Latvakoski

UbiSoft – pervasive software

VTT TIEDOTTEITA – RESEARCH NOTES 2238

UbiSoft - pervasive software

Päivi Kallio, Eila Niemelä & Juhani Latvakoski
VTT Electronics

ISBN 951–38–6452–9 (soft back ed.)
ISSN 1235–0605 (soft back ed.)

ISBN 951–38–6453–7 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0865 (URL: http://www.vtt.fi/inf/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Marja Kettunen

Otamedia Oy, Espoo 2004

Kallio, Päivi, Niemelä, Eila & Latvakoski, Juhani. UbiSoft - pervasive software [Läsnä-älyn ohjelmistojen haasteet ja teknologiat]. Espoo 2004. VTT Tiedotteita – Research Notes 2238. 68 p.

Keywords ubiquitous software, pervasive computing, ubiquitous computing, ubiquitous business

Abstract

Ubiquitous computing enhances computer use by making many computers available throughout the physical environment, while making them effectively invisible to the user. Ubiquitous computing can be seen as a prerequisite for pervasive computing that emphasizes mobile data access, and the mechanisms needed to support a community of nomadic users. Ambient intelligence focuses on a smart way to use communication technology for making life simpler, more enjoyable and interesting. Ubiquitous software is software required for ubiquitous computing environments and in this report it includes pervasive software. Ambience intelligence is out of the scope of this report.

The aim of this report is to offer Finnish companies and preparators of the Tekes-programs (ELMO, NETS, FENIX) a total view of the maturity, development needs and business opportunities of software engineering in the ubiquitous computing area. This report illustrates the state-of-the-art and requirements of ubiquitous software based on recent surveys. This report also provides a view on the state-of-the-practice of ubiquitous software in Finnish companies based on interviews made in some Finnish companies and replies received to a questionnaire sent to bigger sample of companies.

State-of-the-art visions set about global ubiquitous systems, products' short-time-to-market and quality requirements that are tightening up, set ubiquitous software a huge set of requirements that include interoperability, hetereogeneity, mobility, security, adaptability, ability of self-organization, augmented reality and scalable content. Enabling technologies of ubiquitous software are standards, reference architectures and generic software technologies. Ubiquitous software develoment requires applying suitable software architectures and development methods that are presented in this research.

Based on the stat-of-the-practise this report presents that the main challenges of ubiquitous software are achieving adaptable middleware and interoperability between services and networks, developing the required enabling technologies, defining value chain for providing the services and guaranteeing secure transactions between different stakeholders. Ubiquitous computing is seen to combine hardware and software, so new kind of development methods and architectural models are required for ubiquitous service development. Companies will have business opportunities in ubiquitous busi-

ness in middleware components, for example, concerning security, enabling technologies, ubiquitous components and sensors implemented locally to various conditions such as to the surface of paper, small-sized applications, and personalized services.

In principal, we suggest as important research topics in ubiquitous software arena security, management of changing requirements, middleware standards and services, cost efficient architecture solutions and ubiquitous business value chains.

Recommended research topics of ubiquitous software.

	Authors' view	**Companies' view**
Generic	– Security and privacy protection – Ontology-orientated design – Methods and tools for describing service and content semantics	– Data security, ownership and control – Information semantics – User interfaces and interaction of a user in different devices
Software	– Cost-efficient and dynamic networking and architecture solutions – Management of changing requirements – Dynamic architectures – Adaptive middleware services – A unified middleware standard – Service evaluation – Methods and tools for evaluating execution qualities – Methods how to add value to existing software solutions – Methods, tools and platforms for testing embedded product software	– Management of rapidly changing requirements – Defining software-intensive, tailorable, platform-independent product applications – Configuration of the functionality – Dynamic architectures – Middleware and use cases of ubiquitous applications – Component-based design – Defining standardized interfaces
System	– Interoperable distributed software platforms	– Defining services that are adaptable to changes in transmission networks, service level, terminal etc.
Business	– Business value chains	– Defining challenges and costs of the infrastructure implementation – Defining cost-effective and reasonable value chain – Defining consumer needs and potential markets; the needs could be collected with the aid of pre-study of potential applications and their advantages

Kallio, Päivi, Niemelä, Eila & Latvakoski, Juhani. UbiSoft - pervasive software [Läsnä-älyn ohjelmistojen haasteet ja teknologiat]. Espoo 2004. VTT Tiedotteita – Research Notes 2238. 68 s.

Avainsanat ubiquitous software, pervasive computing, ubiquitous computing, ubiquitous business

Tiivistelmä

Läsnä-älyn ohjelmistot lisäävät tietokoneiden käyttöä ja tarjoavat ohjelmistoilla tuotettuja palveluja käyttäjille hyödyntämällä käyttäjien normaalia fyysistä toimintaympäristöä, mutta kätkemällä tietokoneiden olemassaolon itse käyttäjiltä. Verkotettuja sulautettuja järjestelmiä tarvitaan, jotta käyttäjä saa haluamansa palvelut kaikissa mahdollisissa tilanteissa ja ympäristöissä. Tietokoneiden ja tietotekniikan leviäminen edellyttää myös tiedon saatavuutta liikkuvien päätelaitteiden kautta, mikä puolestaan edellyttää erityisiä liikkuvaa käyttäjää tukevia ratkaisuja. Älykkäät ympäristöt pyrkivät hyödyntämään tietoliikenneteknologiaa ihmisen elämän helpottamiseksi ja rikastuttamiseksi. Läsnä-älyn ohjelmisto tarkoittaa verkotettujen ja ympäristöön sulautettujen järjestelmien ohjelmistoa. Läsnä-älyn ohjelmistojen kehittäminen korostaa joko tietokonetekniikkaa tai ihmiskeskeisyyttä, joita molempia asioita on käsitelty tässä raportissa rinnakkain. Älykkäiden ympäristöjen kehittämiseen liittyvät teknologiat on jätetty raportin aihepiirin ulkopuolelle.

Tämän raportin tarkoitus on tarjota suomalaisille yrityksille ja Tekes-ohjelmien (ELMO, NETS, FENIX) valmistelijoille kokonaisnäkemys läsnä-älyn ohjelmistoteknologioiden kypsyydestä, kehitystarpeista, liiketoimintamahdollisuuksista ja ohjelmistokehityksen verkottumisesta sekä läsnä-älyn sovellusten mahdollisuuksista tulevaisuudessa. Raportti kuvaa läsnä-älyn ohjelmistojen teknologialle asettamat vaatimukset ja teknologioiden kypsyyden perustuen tuoreisiin tutkimustuloksiin. Raportti esittää myös läsnä-älyn sovelluksia kehittävien suomalaisten teollisuusyritysten käsityksen teknologian nykytilasta haastatteluihin ja kyselytutkimukseen perustuen.

Globaaleista langattomista kommunikointijärjestelmistä esitetyt visiot, palvelutuotteille asetettava lyhyt kehitysaika ja yhä kiristyvät laatuvaatimukset asettavat sekä järjestelmille että ohjelmistoille suuren joukon haasteita kuten liikkuvuus, yhteistoiminnallisuus ja mukautuvuus. Standardit, viitearkkitehtuurit ja yleiset ohjelmistoteknologiat ovat esimerkkejä teknologioista, jotka mahdollistavat läsnä-älyn ohjelmistojen kehittämisen. Koska kaikkialle leviävä tietojenkäsittely yhdistää ohjelmistot ja laitteistot, läsnä-älyn ohjelmistojen kehittämiseen tarvitaan uudenlaisia menetelmiä ja arkkitehtuurimalleja.

Tässä raportissa esitetään, että läsnä-älyn ohjelmistojen toteuttamisen suurimpia haasteita ovat mukautuvan välitason ohjelmistojen kehittäminen, ohjelmistopalvelujen ja kommunikointiverkkojen yhteistoiminnallisuuden aikaansaaminen, vaadittavan

infrastruktuurin kehittäminen, toimivan arvoketjun määritteleminen palvelujen tuottamiseksi ja palvelujen turvallisen toimituksen takaaminen eri osapuolten välillä. Yrityksillä on liiketoimintamahdollisuuksia läsnä-älyn ohjelmistojen ja järjestelmien alueella mm. välitason komponenteissa, palvelut mahdollistavissa teknologioissa, sulautettujen järjestelmien komponenteissa, antureissa ja pienikokoisissa henkilökohtaistetuissa palveluissa.

Suosituksia läsnä-älyn ohjelmistojen tutkimusaiheiksi.

	Kirjoittajien näkökulma	**Yritysten näkökulma**
Yleiset teknologiat	– tietoturva ja yksityisyyden suojaus – ontologia perusteinen suunnittelu – menetelmät ja työkalut palvelun ja sisällön merkityksen kuvaamiseksi	– tietoturva, omistusoikeudet ja niiden hallinta – tiedon semantiikka – käyttöliittymät ja erilaiset vuorovaikutustekniikat
Ohjelmistot	– kustannustehokkaat dynaamiset välitason palvelut ja arkkitehtuuriratkaisut – nopeasti muuttuvien vaatimusten hallinta – dynaamiset arkkitehtuurit – adaptiiviset välitason palvelut – yhdistetty välitason standardi – palvelujen varmentaminen – menetelmät ja työkalut suoritusaikaisten laatutekijöiden varmentamiseksi – menetelmät olemassa olevien palvelujen sovittamiseksi läsnä-älyn sovelluksiksi – sulautettujen ohjelmistotuotteiden kehittämistä tukevat menetelmät, työkalut ja alustat	– nopeasti muuttuvien vaatimusten hallinta – dynaamiset arkkitehtuurit – ohjelmistokeskeisten, räätälöitävien ja alustariippumattomien sovellusten kehittäminen – toiminnallisuuden konfiguroitavuus – läsnä-älyn sovellusten välitason palvelut ja käyttötapaukset – komponenttiperusteinen suunnittelu – standardiliityntöjen määrittely
Järjestelmä	– yhteentoimivat hajautetut ohjelmistoalustat	– tiedonsiirtoverkoissa, palvelutasoissa ja päätelaitteissa tapahtuviin muutoksiin sopeutuvien palvelujen määrittely
Liiketoiminta	– liiketoiminnan arvoketjujen määrittely	– infrastruktuurin haasteiden ja kustannusten määrittely – kustannustehokkaiden ja järkevien arvoketjujen määrittely – asiakas- ja markkinatarpeiden määrittely; tarpeiden tunnistamiseksi aktivoidaan sovellusten mahdollisuuksia ja hyötyjä kartoittavia esitutkimuksia

Preface

The background of this report is the need presented by Tekes and Finnish companies to clarify the development needs in the ubiquitous area from the software point of view. This need is based on VTT Electronics' Ubicom-raport "Ubicom applications and technologies" (Ailisto et al. 2003). The aim of this report is to offer Finnish companies and preparators of the Tekes-programs (ELMO, NETS, FENIX) a view from the maturity, development needs, business opportunities and networked software development in ubiquitous computing area.

The work has been carried out within the Opera project of VTT Technical Research Centre of Finland and has been funded by Tekes, the National Technology Agency of Finland. This report is based on the state-of-the-art surveys and the interviews and questionnaire done for industrial companies. The state-of-the-art survey of this report was performed by collecting the most recent surveys made in several research projects and the state-of-the-practice by interviewing some Finnish companies autumn, 2003 and sending them thereafter a questionnaire to wider amount of companies.

Prof. Eila Niemelä functioned as responsible manager of this report. State-of-the-practice of this report was realized by senior research scientists Päivi Kallio and Juhani Latvakoski. Eila Niemelä has contributed to all other chapters except 4 of this report, Juhani Latvakoski to Chapter 2 and the testing in Chapter 3, and Päivi Kallio to Chapters 1, 4 and 5 and the editing of this report. Matti Sihto from Tekes provided his valuable comments and views for this report.

We would like to sincerely thank the representatives of the companies that participated in this research for their valuable support.

Oulu, March 2004

Päivi Kallio Eila Niemelä Juhani Latvakoski

Contents

Abbreviations

3GPP	3rd Generation Partnership Project
2G,3G,4G	2nd Generation, 3rd Generation, 4th Generation (mobile phone networks)
ACD	Application customized description
AIMS	Adaptive Introspective Management System
AOP	Aspect Oriented Programming
API	Application Programming Interface
ASD	Adaptive Software Development
AWG	Architecture Working Group
CAST	Computer Aided Software Testing
C/S	Client Server
CFS	Context File System
CM	Coordination Medium
COM(+)	Component Object Model (Microsoft)
CORBA	Common Object Request Broker Architecture (OMG)
CPU	Central Processing Unit
DCOM	Distributed COM (Microsoft)
DRM	Digital Rights Management
EJB	Enterprise Java Beans
ETSI	European Telecommunications Standards Institute
EU	European Union
FDD	Feature-Driven Development
FIPA	Foundation for Intelligent Physical Agents
GMA	Grid Monitoring Architecture
GPRS/ EDGE	General Packet Radio Service/ Enhanced Data rates for Global Evolution
GSP	Generic Switch Place
HAVi	Home Audio / Video Interoperability
HomeRF/ SWAP	Home Radio Frequency/ Shared Wireless Access Protocol
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
HTTPR	Hypertext Transfer Protocol Reliable
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMPP	Instant Messaging and Presence Protocol
IP	Internet Protocol
IPv6	IP version 6
IS	Internal Structure

IT	Information Technology
ITEA	Information Technology for European Advancement
ITU	International Telecommunication Unit
MARS	Mobile Agent Reactive Spaces
MDA	Model Driven Architecture
MIP	Mobile Internet Protocol
MITA	Mobile Internet Technical Architecture
MOF	Meta Object Facilities
MOM	Message Oriented Middleware
MPU	Message Processing Units
MQ	Messaging and Queuing
MS	Microsoft
MSP	Message-passing Switch Place
MVC	Model-View-Controller
NetBSD	Net Berkeley Software Distribution
OBSAI	Open Base Station Architecture Initiative
ODP	Open Distributed Processing
OMA	Open Mobile Alliance
OMG	Object Management Group
OSA	Open Service Architecture
OSGi	Open Services Gateway initiative
P2P	Peer-to-Peer
PAC	Presentation-Abstraction-Control
PAN	Personal Area Network
PDS	Proactive Directory Service
PKI	Public Key Infrastructure
QoS	Quality of Service
RPC/RMI	Remote Procedure Call/ Remote Method Invocation
SA	Survival by Adaptation
SIP	Session Internet Protocol
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SP	Survival by Protection
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/ IP
TPA	Trading Partners Agreement
TTCN	Tree and Tabular Combined Notation
UDDI	Universal Description, Discovery and Integration
UDP/IP	User Datagram Protocol IP
UML	Unified Modeling Language
W3C	World Wide Web Consortium

WML	Wireless Markup Language
VFS	Virtual File System
VHE/OSA	Virtual Home Environment/ OSA
WLAN	Wireless Local Arean Network
WSCL	Web Services Conversation Language
WSDL	Web Service Definition Language
WSFL	Web Services Flow Language
WSXL	Web Service Experience Language
WWRF	Wireless World Research Forum
xDSL	Digital Subscriber Lines
XHTML	eXtensible HTML
XMI	Meta Data Interchange
XML	eXtensible Markup Language
XP	eXtreme Programming

1. Introduction

1.1 Background and aims

The background of this report is the need presented by Tekes and Finnish companies to clarify the development needs in the ubiquitous area from the software point of view. This need is based on VTT Electronic's Ubicom-raport "Ubicom applications and technologies" (Ailisto et al. 2003). The aim of this report is to offer Finnish companies and preparators of the Tekes-programs (ELMO, NETS, FENIX) a view of the maturity, development needs, business opportunities and networked software development of software engineering in the ubiquitous computing area.

This report aims to present the needs for ubiquitous software and process development as a whole. The aim of the evaluation is to illustrate the main challenges in ubiquitous software development, enabling and enhanced software technologies and their maturity level. The report explores the following topics:

- requirements of ubiquitous software development,
- architectural solutions and standardization of the area,
- the main actors in the area,
- estimation of the future evolution of the area in a 5-10 years period,
- the opportunities for Finnish industry, and
- recommendations for required actions.

The state-of-the-art survey of this report was done by collecting the newest surveys made in several research projects concerning software architectural models and development methods of pervasive software. The state-of-the-practice was realized during autumn 2003 by interviewing experts from eight Finnish companies and sending a questionnaire to 80 Finnish companies. The results of the work reported in this research note were completed in 2004.

1.2 Definition of concepts

Ubiquitous computing enhances computer use by making many computers available throughout the physical environment, while making them effectively invisible to the user (Weiser 1991; Weiser 1993). Mark Weiser from Xerox PARC expressed the goal as to achieve the most efficient kind of technology that is essentially invisible to the user, to make computing as ordinary as electricity. Ubiquitous computing embeds com-

puter technology in our every-day environment providing humans with information services and applications through any device over different kinds of networks. Thus, computer technology is addressed by adding computers everywhere. At the beginning of the ubiquitous computing efforts, the focus was on small special purpose devices, network protocols, interaction substrates, and new styles of applications. After the first prototypes, additional research directions were identified:

- wireless communications,
- partitioning and disconnected operation,
- location and resource discovery,
- privacy, and
- power consumption.

Ubiquitous software is software required for ubiquitous computing environments. The other terms, used in the same context but from different points of view, are pervasive computing and ambient intelligence. *Pervasive computing* emphasizes mobile data access and the mechanisms needed to support a community of nomadic users, smart or “active" spaces and context awareness. Pervasive computing also emphasizes the way people use devices to interact with the environment, the best ways to deploy new functions on a device and exploit interface modalities for specific tasks. There are three major focus areas in pervasive computing. First is the way people view mobile computing devices and use them within their environments to perform tasks. Secondly, the way applications are created and deployed to enable such tasks to be performed, and thirdly, how the environment is enhanced by the emergence and ubiquity of new information and functionality (Raatikainen et al. 2003).

Ambient intelligence focuses on a smart way to use communication technology for making life simpler, more enjoyable and interesting. Thus, ubiquitous computing is a prerequisite for pervasive computing, whereas ambient intelligence is based on pervasive computing technology that is enhanced by intelligent user interfaces suitable for different usage contexts (Purhonen & Tuulari 2003). In ambient intelligence persons are surrounded by intelligent interfaces supported by computing technology, which is everywhere, embedded in clothes, furniture, walls, vehicles etc. As interaction, one uses the whole body: speech, pointing, gestures and even direction of sight. The intelligent environments will play a role in all activities: from houses to industrial plants and healthcare. The concept ”ambient intelligence” is not limited to services or devices, but extends to any kind of software system. In this research ubiquitous software includes also pervasive software. Ambience intelligence is out of the scope of this report.

1.3 Overview of this report

This report focuses on the ubiquitous software technologies so that Chapter 2 illustrates the requirements of ubiquitous computing. Chapter 3 provides a state-of-the-art of ubiquitous software. Chapter 4 describes the summary and analysis of the interviews of Finnish companies and the replies received to the questionnaire sent afterwards to wider amount of Finnish companies. Chapter 5 illustrates the actors in ubiquitous software development. Chapter 6 draws conclusions, defines the opportunities of Finnish companies in the ubiquitous arena and gives recommendations for future research activities.

2. Requirements for ubiquitous software

Figure 1 visualizes the requirements of ubiquitous computing that arise from visions of ubiquitous computing. The system, software and business are affected by these requirements. In Figure 1 the system refers to the computing platforms, software to software implemented as services and components of the system, and business to the network of actors providing added value and components and services to the system development and the process for producing the system and its components (i.e. development view). The main requirements of ubiquitous computing are interoperability, heterogeneity, mobility, adaptability, security and privacy, self-organisation, and last but not least augmented reality and content scalability. The aim of this chapter is to provide some more details for clarification of Figure 1.

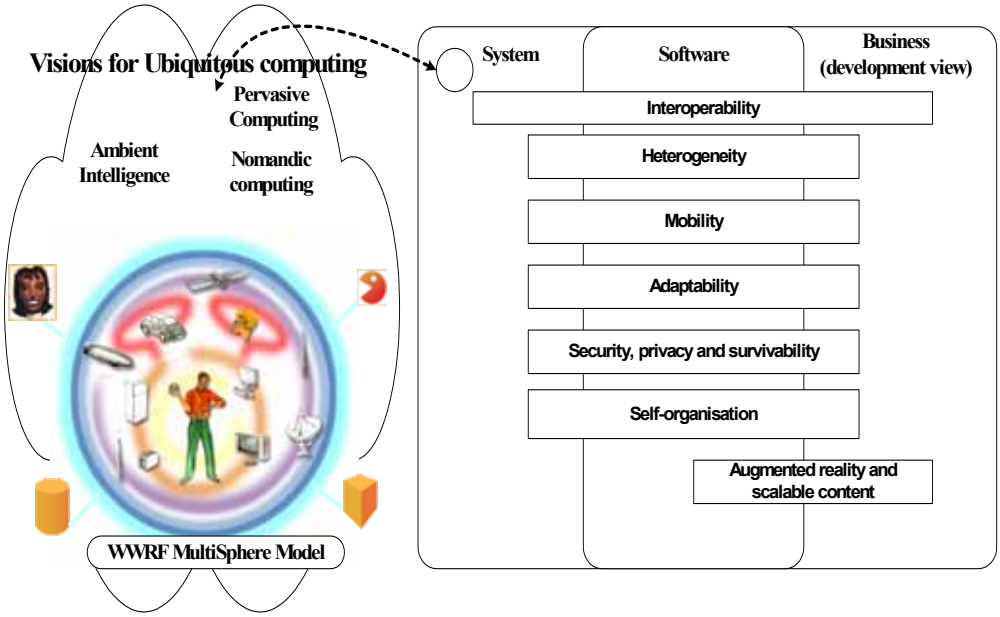

Figure 1. Requirements architecture for ubiquitous computing.

2.1 Visions of ubiquitous computing

The notion of *ubiquitous computing* is related to a set of other paradigms such as nomadic computing, pervasive computing and ambient intelligence. A more concrete view (based on communication technologies) of the future wireless system is Wireless World Research Forum's (WWRF) MultiSphere model, which consists of several levels: Personal area network (PAN), immediate environment, instant partners, radio accesses, interconnectivity, and cyberworld (WWRF 2001). The model is "I" centric in the sense, that everything starts from a human described in the center of the model. The closest interaction will happen with the PAN elements that are nearest to us or might even be

part of our body. The immediate environment refers to the real world elements that surround us. Today, we usually do not interact with the elements in an electronic way, but in the future this is expected to happen. In addition, future interaction should even personalize the devices in the immediate environment. For example, TV sets should know the programs the user is interested in. A toaster might want to deliver toast just as a person wants it, and a fridge wants to tell what we could have for breakfast or propose that we go and buy milk.

Instant partners refers to the more complex elements around us, such as cars. It is expected that we want to talk with them or relay information through them. The same applies to the people around us, which enables a new form of networking people with whom we want to be closely interconnected, e.g. local chat communities.

Efficient radio accesses that need to be publicly accessible for any entity are the essential fundamental requirement to enable ubiquitous coverage of a wide area system. Current infrastructures will be enhanced by novel solutions such as mobile base stations. The system needs to be adaptive to various user devices, equipped with simple interaction with a network having as low operational cost as possible. Another essential issue is the universal wireless interconnectivity, which can be claimed to increase the value of communications proportionally to the square of the number of the connected devices. One key requirement is offering the right level of support for the various specialized radio interfaces and terminals.

Ubiquitous computing environments also require adaptive interaction with the radio technology and applications. End to end support for the required quality of service level for interconnectivity is essential. The cyberworld is enhancing our physical presence in the virtual world that is already visible in the form of “web” services and advanced games. In the future, we can stay in touch with our (semantic) agents, knowledge bases, communities, services and transactions.

Ubiquitous wireless world systems trigger a huge set of requirements for both system and software technologies. One discussion is provided in the roadmap of future telecommunication technologies (Sipilä 2002), in which the future requirements are discussed in terms of inter-networking, micromechanical radio frequency systems, service architectures and smart human environments. A ubiquitous computing environment requires an advanced computing platform with energy efficient radio accesses. In addition, it can be assumed that one of the most essential challenges will arise from the huge complexity and intelligence required in software that embeds the brains of the system.

2.2 Some essential requirements

This section describes some essential requirements from the software and software development points of view as presented in Figure 1.

2.2.1 Interoperability

Interoperability is one of the most essential requirements of ubiquitous software, because the need to network the embedded products of different vendors with cyber world applications is increasing in the ubiquitous environment. The increasing amount of microprocessors and their networking highlights the need for distributed software platforms applicable for distributed wireless computing (Coulouris et al. 2001), de facto or de jure standards (consensus between different actors), and especially testing that software implementations realize these standards correctly.

Integrability is related to interoperability and interconnectivity. *Interoperability* is a sub-characteristic of integrability and is partially defined by *interconnectivity*, the ability of software components to communicate and exchange information. Thus, interconnectivity is a prerequisite for interoperability, the ability of software to use the exchanged information and provide something new originating from exchanged information. Interconnectivity and interoperability are execution qualities, whereas integrability has a larger scope, impacting the development and evolution of a system. Therefore, integrability is to be considered together with the features of a product family, domain requirements, coarse-grained architectural elements and the ways to develop and maintain software systems. Interoperability is considered when components and their interactions are defined in detail and finally observed as executable models, simulations and running systems. (Taulavuori et al. 2004)

Integration architecture is a software architectural description of the overall solution to interoperability problems between various component systems. The focus is on pre-integration assessment that tries to discover inherent interoperability problems based on the architecture mismatch analysis that describes the underlying reasons for interoperability problems among software components (Garlan et al. 1995). Software components should be built to be independent of the context in which they are used, as this allows their use in different computing environments and applications. Component interfaces should be specified by a uniform description language and thereafter the component specifications can be used for binding components dynamically, i.e. when the context of a component changes, the interconnections between components are changed according to the new context. (Niemelä &Vaskivuo 2004)

2.2.2 Heterogeneity

In the future, ubiquitous world heterogeneous networks will be integrated. The devices connected by the heterogenous networks will have different screen resolutions, user interaction methods, radio capabilities, memory, power and processing capabilities, as well as mobility. Services can be accessed through heterogeneous networks with widely varying transport capability, quality and usage cost and they may have different requirements for bandwidth, real-time capabilities as well as input output methods. The user interaction with computers refers to various facilities and mechanisms that enable users to interact with devices using multiple methods (multi-modality) such as speech, hand movement, screens, buttons etc. This kind of interaction requires novel solutions in the form of embedded sensors.

In software, heterogeneity is expressed by a diversity of software structures, component models, interface technologies and languages. The component model depends on the selected middleware that sets requirements and constraints for the architectural structure and, in many cases, also for the implementation languages used in application development.

The infrastructure of different networks is provided by multiple actors. In addition, services may consist of components and subsystems provided by different actors. The value chain may be complex network (value network) between these different actors.

2.2.3 Mobility

Management of mobility will be an important characteric of a ubiquitous system. There are, however, different kinds of mobility schemes, such as terminal mobility, personal mobility, session mobility and service mobility. Users shall be supported in such a way that they can move from one place or terminal to another and still get a personalized service (Schulzrinne & Rosenberg 2000). In the future, the networks may also be mobile and dynamic, and therefore, full mobility is an essential requirement in the ubiquitous wireless world (Latvakoski et al. 2004).

From a software perspective, mobility can be divided into actual, virtual and physical mobility (Cabri et al. 2002). Actual mobility is an extension to the capability of an autonomous software agent that dynamically transfers its execution, i.e. its code, data, and execution state, towards the nodes where the resources it needs to access are located. Exploitation of actual agent mobility can save network bandwidth and increase the reliability and efficiency of the execution. Agent mobility can effectively be exploited to have nomadic users assisted by personal software agents capable of following them in their activities. Virtual agent mobility is the ability to be (network-) aware of the multiplicity of networked execution environments (e.g. Internet nodes or administra-

tive domains of nodes). When agents are aware of the distributed nature of the target, and explicitly locate and access Internet resources in the environment, it is a kind of virtual mobility of agents across execution environments. Physical agent mobility means mobile and wireless computing devices connecting to the Internet from dynamically changing access points. An active space extends the physical space, i.e. physical objects, networked devices, and users with well-defined physical boundaries, adding coordination via a context-based software infrastructure (Roman et al. 2002). (Niemelä & Vaskivuo 2004)

2.2.4 Security, privacy and survivability

In ubiquitous services the security mechanisms should support authentication, authorization, confidentiality, reliable transactions, and privacy of communication and content, end systems and users' location, and protection against denial of service attacks (Sipilä 2002).

Software engineers define survivability as the ability of a system to fulfill its mission in a timely manner, and also in the presence of attacks and failures. Thus, survivable systems require self-healing infrastructure for distributed applications with improved qualities such as security, performance, reliability, availability and robustness. (Amin 2000) A key characteristic of a survivable system is its capability to deliver essential services even in the face of attack, failure or accident. These services are delivered with the specified levels of integrity, confidentiality and performance. Thus, it is important to define minimum levels of quality attributes that must be associated with the essential services. (Tarvainen 2004) These qualities are so important that survivability is often expressed in terms of trade-offs among multiple quality attributes such as performance, security, reliability, availability, and modifiability. Because quality attributes represent broad categories of related requirements, a quality attribute may contain other quality attributes (Ellison et al. 1999; Matinlassi & Niemelä 2003).

Current facets of dependability, such as reliability and availability, do not address the needs of critical information systems because they do not include the notion of a degraded service as an explicit requirement. A precise notion is needed for the forms, in which a degraded service is acceptable, under what circumstances each form is most useful and the fraction of time a degraded service is acceptable. This concept is termed survivability; a new branch of dependability. (Knight & Sullivan 2000; Knight et al. 2003)

Survivability requirements can vary substantially depending on the scope of a system and the consequences of the failure and interruption of a service. The definition and analysis of survivability requirements is the first step in achieving system survivability.

Survivability must address not only the functional requirements of software, but also the requirements for software usage, development, operation and evolution (Tarvainen 2004). Five types of requirements definitions are relevant to survivable systems (Ellison et al. 1999): System/Survivability Requirements, Usage/Intrusion Requirements, Development Requirements, Operations Requirements, and Evolution Requirements.

Survivability is not synonymous with fault tolerance. Fault tolerance is a technique for achieving a certain dependability property. In terms of dependability, it is referred to a system as reliable, available, secure, safe, etc., or some combination of them using the appropriate formal definition. A fault tolerant system really means a statement about the system's design (Knight & Sullivan 2000). Fault tolerance enables systems to continue to provide services in spite of the presence of faults. Fault tolerance consists of four phases: error detection, damage assessment, state restoration and continued service. The first two phases constitute comprehensive error detection and the latter two phases constitute comprehensive error recovery. Survivability is intimately related to and dependent upon both the recognition of certain system faults that affect the provision of a service (error detection) and the proper response to these system faults in order to provide some form of continued service (error recovery). (Elder 2001)

It is important to understand the relationship between survivability and security. An application may employ security mechanisms such as passwords and encryption and may still be fragile, for instance, by failing when a server or a network link dies. On the other hand, a survivable application must be able to survive malicious attacks, and therefore survivability must involve security. In that case, there are two kinds of survivability: survival by protection (SP) and survival by adaptation (SA). In SP, security mechanisms like access control and encryption attempt to ensure survivability by protecting applications from harmful (accidental or malicious) changes in the environment. In SA, the application can survive by adapting itself to the changing conditions. These two kinds of survivability are not mutually exclusive: an application may utilize security mechanisms in SA as well. For example, it may start using access control or increase the key length when it perceives the threat of an intrusion. (Pal et al. 2000; Tarvainen 2004)

2.2.5 Adaptability

Services must adapt to different kinds of terminals and networks, as well as handle dynamically emerging and evolving contexts and users preferences. The system may have different kinds of radio capabilities (multiradio), and due to mobility, dynamically changing conditions make adaptability anything but an easy challenge.

Application-aware adaptation that is part of context-awareness means collaboration between the system infrastructure and individual applications (Noble et al. 1997). The system manages the resources; it monitors resource levels, notifies applications of relevant changes, and enforces resource allocation decisions. Each application independently decides how to adapt when notified. Resource management is centralized (and embedded to the middleware) but adaptation is controlled in a decentralized way. This is a mixed controlling architecture; centralized monitoring and tracking, and decentralized decision-making. (Niemelä &Vaskivuo 2004)

There are many strategies to adapt the applications. In *the laissez-faire approach*, the responsibility of adaptation is left to individual applications and no system support is provided for adaptations. However, this approach lacks the central arbitrator to resolve the incompatible resource demands of different applications and to enforce limits on resource usage. Even though the system support for adaptation can be avoided in this approach, the applications become more difficult to implement and the size of the applications increases because each application needs to implement its own adaptation functionality individually. (Noble & Narayannan 1997 et al.)

The other extreme of adaptation strategies is the *application-transparent approach*, where adaptation does not require any changes in applications, but it is left fully to the responsibility of the underlying system. Even when providing backward compatibility with existing applications, this approach has drawbacks. There may be situations where the adaptation performed automatically by the system is inadequate or even harmful.

Between these two extremes of adaptation strategies lie a number of solutions that are collectively referred to as *application-aware adaptation*. Application-aware adaptation emphasizes the collaborative partnership of applications and the system in the adaptation functionality. This approach permits applications to determine the best adaptation behavior for the situation, but preserves the ability of the system to monitor resources and enforce allocation decisions. Application-aware adaptation also decreases the application size compared to the laissez-faire adaptation approach, because part of the adaptation functionality is provided by the system and every application does not need to have embedded adaptation functionality. (Pakkala 2004)

Agility means a set of combined quality properties, sensitivity to varying resources (e.g. battery power and bandwidth) and sensitivity to changes in resource availability (e.g. data sharing in intermittent connections) (Noble et al. 1997). Resource need for concurrent applications is changing according to the situation in which services are used and how they are processed. In this case, agility is mapped to the execution of a software system and its ability to manage changes that are unpredictable in terms of time, but whose characteristics are predictable**.** However, in addition to execution agility all soft-

ware systems also embody evolvability that means the ability to handle changes in the long-term, considering the life-cycle of a system. (Niemelä &Vaskivuo 2004)

2.2.6 Ability of self-organization

Self-organization is a process where the organization of a system spontaneously increases, without this increase being controlled by the environment or an encompassing or otherwise external system. A self-organizing system not only regulates or adapts its behaviour, but it also creates its own organization (Heylighen & Gersheson 2003).

Self-organization applies concepts of self-learning, expert systems, chaotic theory, fuzzy logic, etc., to enable more smooth application of computing systems for its users. In addition, self-organization may be applied for communication networks, such as ad hoc networks, to achieve improved performance, efficiency, minimize cost and increase reliability and survivability.

Ad hoc networks refer to the system that consists of devices dynamically connected to each other using wireless media (Perkins 2001; Toh 2002). Ad hoc networks are automatically organized without any static configuration or centralized management (self-organization of communication networks). From a user's perspective these systems can be called spontaneous systems (Latvakoski et al. 2004, Kindberg & Fox 2002). From a software perspective, ability of self-organization refers to the ability to dynamically reorganise the structure of software, i.e. dynamic software architectures.

2.2.7 Augmented reality and scalable content

The increasing amount of information (content), content and service providers and network services is making life more difficult for humans. Additionally, new ways to look at the content like an augmented reality are emerging. In augmented reality human awareness is augmented by using virtual context in parallel with human-sensed context. One example of this kind of solution is described in Antoniac et al. (2002). Therefore, the requirements for augmented reality and scalable content include many perspectives such as e.g. security, privacy protection, Digital Rights Management (DRM), adaptability, self-organization and semantic awareness.

The term *fidelity* has been used for the property of a system that defines the degree to which data presented at a client matches the reference copy at the server (Noble et al. 1997). Fidelity includes three dimensions: consistency, the type of data and tradeoffs made by applications. When network connectivity is poor or nonexistent, data provided to applications may be stale but still useful for achieving appropriate functionality in a

system. Data *consistency* means high availability of shared data in intermittent networked systems. The data types are based on time, state and frequency. Sampling rate and timeliness are quality properties of the type of telemetry data. On the other hand, the size and resolution of data is considered as the fidelity of spatial data, e.g. topographical maps. In addition to the image quality of each frame, frame rate is a key issue in video streaming. Applications that use data make different tradeoffs among the dimensions of fidelity, the quality property of shared data. (Niemelä & Vaskivuo 2004)

2.3 Summary

Table 1 below summarizes the requirements of computing and their effect on system, software and business. The state-of-the-art of ubiquitous software technologies described in the next chapter tries to answer how ubiquitous computing requirements are met by software technologies.

Table 1. Summary of ubiquitous computing requirements

Requirement	System	Software	Business (development view)
Interoperability	• Increasing amount of microprosessors and their networking	• Distributed software platforms • Integrability and interoperability • Software Testing	• Multiactor system infrastructure • Multiprovider and multiactor services • Value network
Heterogeneity	• Integration of heterogeneous networks • Various device capabilities • Various user interaction methods	• Service access via different network infrastructures • Services have different needs for the network and device capabilities • User interaction support	• Multiactor system infrastructure • Integration of multiprovider and multiactor services • Value network
Mobility	• Physical device, personal, session, service mobility • Full mobility	• Actual, virtual and physical mobility • Physical device, personal, session, service mobility • Full mobility	• Roaming in multiactor system • Value network
Security, privacy and survivability	• Security (authentication, authorization, confidentiality, accountability) • Reliable transactions • Privacy of communication and content, • Terminal and user location.	• Security • Reliability • Survivability • Location-monitoring	• Security and fault tolerance of multiactor system infrastructure • Security and privacy of multiprovider service network
Adaptability	• Different kinds of terminals and networks • Dynamically changing contexts and users preferences • Different kinds of radio capabilities (multiradio), and dynamically changing conditions	• Context awareness • Personalization • Resource management • Agility	• Multiactor system infrastructure • Multiprovider and multiactor services
Self-organization	• Ad hoc networking	• Smooth application of computing platforms for users • Spontaneity • Dynamic structures	• Multiactor system infrastucture • Multiprovider and multiactor value network
Augmented reality and content scalability	• Content networking	• Semantic awareness • Scalability, augmented reality • Fidelity	• Multiprovider and multiactor services/content value network

3. State-of-the-art in ubiquitous software technologies

This chapter captures the state-of-the-art of ubiquitous software technologies considering standards and models that can be used to guarantee that ubiquitous software meets the special requirements mentioned in the previous chapter. This chapter also illustrates some new development approaches that are applicable in the development of ubiquitous and pervasive software systems.

Ubiquitous computing means highly distributed systems that consist of applications, middleware and system infrastructure software. Middleware embodies a variety of distributed computing services and application development supporting environments that operate between the application logic and the underlying system infrastructure software (Charles 1999). The important capabilities of software in a pervasive computing environment are its ability to provide interoperability for heterogeneous systems that accommodate to dynamically changing resources. Resource variability arises through the user's changing needs, user mobility and through the need to exploit time-varying resources of the environment, e.g. wireless bandwidth. This means that devices are made aware of their contexts, the habits of a device user, the status of the physical and social surroundings of the user and the status of the device itself. Thus, context awareness is the capability of pervasive middleware to handle unpredictable changes.

Figure 2 represents some enabling and advanced software technologies mapped to the architectural layers of ubiquitous systems. Enabling technologies are based on standards, reference architectures and generic software technologies used for wireless and wired ubiquitous systems. Advanced software technologies are micro architectures, adaptive supporting services, special interface standards and specification languages used in defining software services. Advanced software technologies are based on enabling technologies commonly applied in embedded systems. Some of them are mature technologies, used for years, others are emerging and under continuous technology development. For example, GPRS (General Packet Radio Service) and ODP (Open Distributed Processing) are mature technologies, whereas 4G (4^{th} Generation of Mobile phone networks) and design patterns of wireless service engineering are technologies under active development. However, the trend is to adopt and adapt existing technologies to pervasive computing environments, and to develop new software technologies (i.e. languages, supporting services and concepts) that help in the development of ubiquitous software services for end-users. Figure 2 is not exhaustive but tries to illustrate the relationships between enabling and advanced technologies applicable for ubiquitous software. The aim is to give an overview of the technologies that will be introduced next.

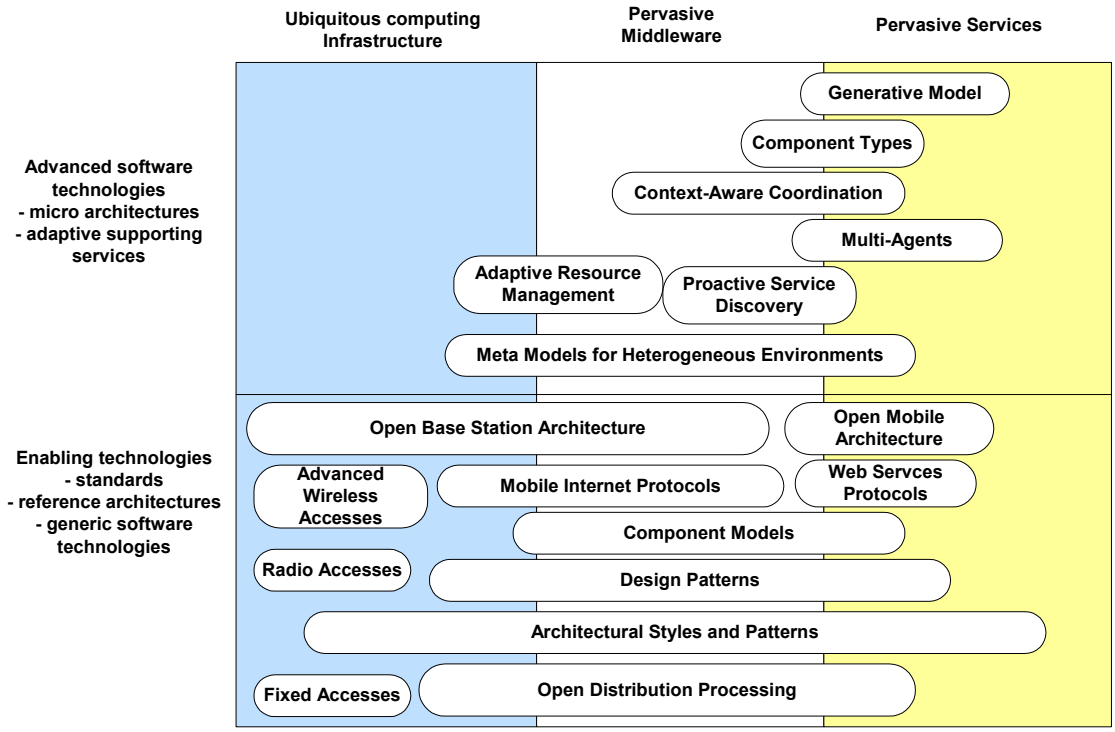

Figure 2. Examples of software technologies of ubiquitous systems.

3.1 Enabling technologies

Enabling technologies of heterogeneous networks that constitute the basis of service architectures of ubiquitous computing environments are classified into four categories:

- generic software technologies,
- application level technologies,
- service level technologies, and
- infrastructure level technologies.

Generic software technologies concerning software engineering methods, architectural styles and patterns as well as component models and appropriate languages are used at the application level as well as at the middleware and infrastructure level. Generic software technologies give standard-based methods and technologies that all kinds of software should be based on. Object-oriented engineering methods, e.g. UML (Unified Modeling Language), have recently been widely used in industry and thus, they will also be applied in pervasive software development. Programming and interface lan-

guages such as Java, C++, IDL (Interface Definition Language) and XML (eXtensible Markup Language) are also technologies commonly used for defining and implementing software components and services. Interface technologies are even more important in the service development in which software developed in multi-organizational settings are composed and used together. COM+ (Component Object Model) and EJB (Enterprise Java Beans) are the most promising component models that will also be applied to pervasive software development. Architectural styles and patterns such as an implicit invocation style and an observer pattern as well as architectural viewpoints will play a key role in the pervasive service development. QADA[SM1] is an architecture development method that focuses on the quality requirements of the systems and retains them as driving factors for achieving a reusable service architecture that can be shared by a wide community of pervasive software developers.

Profiling and the techniques used in artificial intelligence are examples of enabling technologies for context-aware, personalizable, and adaptive applications. Web technologies like HTML (Hyper Text Markup Language), XHTML (eXtensible HTML), WML (Wireless Markup Language) and XML, WSXL (Web Service Experience Language) are used for adaptive user interfaces.

Distribution architectures such as ODP implemented as middleware technologies like Java RMI, CORBA (Common Object Request Broker Architecture), and DCOM (Distributed COM) are also used in pervasive middleware, as shown in the middleware solutions illustrated later in this report. However, communication of mobile and wireless services is more often based on assynchronous messages than synchoronous procedure calls. Although Message Oriented Middleware (MOM) has not been used as a pervasive middleware, technologies as OSGi (Open Service Gateway initiative) and HAVi (Home Audio/Video Interoperability) are messages-oriented and used in the service archtitecture development. VHE/OSA (Virtual Home Environment/Open Service Architecture), Parlay, JAIN and Jini specify the application interfaces used in service architectures. Security technologies such as SSL (Security Sockets Layer) and PKI (Public Key Infrastructure) as well as session technologies, for example, SIP (Session Initiation Protocol) are also suitable for ubiquitous systems. Although there are great a number of technologies applicable for pervasive and ubiquitous software, there is no unified and explicit view of which technologies are best in which kind of context. There is not either a unified technology map as web services have. Figure 3 represents the web services stack as an example of technology map, where enabling technologies are mapped to each layer of the protocol stack. A similar kind of stack is required for pervasive software devel-

[1] QADA (Quality-driven Architecture Design and quality Analysis) is the service mark of VTT Technical research Centre of Finland.

opment. The technologies mentioned in the web services stack might be suitable for pervasive software based on fixed accesses, but they are not necessarily proper solutions for wearable uqibuitous systems or mobile applications.

There are several enabling technologies for ubiquitous infrastructure services. MIP (Mobile Internet Protocol), IPv6 (Internet Protocol version 6) and Fireware specify mobility technologies and protocols, and communication is based on wireless, cellular, ad hoc and wired networks. Wireless networks can be based on WLAN (Wireless Local Area Networks), Bluetooth or HomeRF/SWAP (Home Radio Frequency/Shared Wireless Access Protocol) technologies. Cellular networks may use the specifications of 2G (2nd Generation), GPRS/EDGE (GPRS/ Enhanced Data rates for Global Evolution), 3G (3rd Generation) or 4G (4th Generation) networks. Wired networks use, for example, Powerline, cable, xDSL (Digital Subscriber Lines), and the Internet over Fibre. Communication technologies are more thoroughly surveyed in (Ailisto et al. 2003).

Besides communication technologies, security technologies and operating systems are infrastructure technologies required in ubiquitous systems. Because ubiquitous systems are heterogeneous systems, the scale of operating systems is wide, from micro kernels to commercial distributed operating systems.

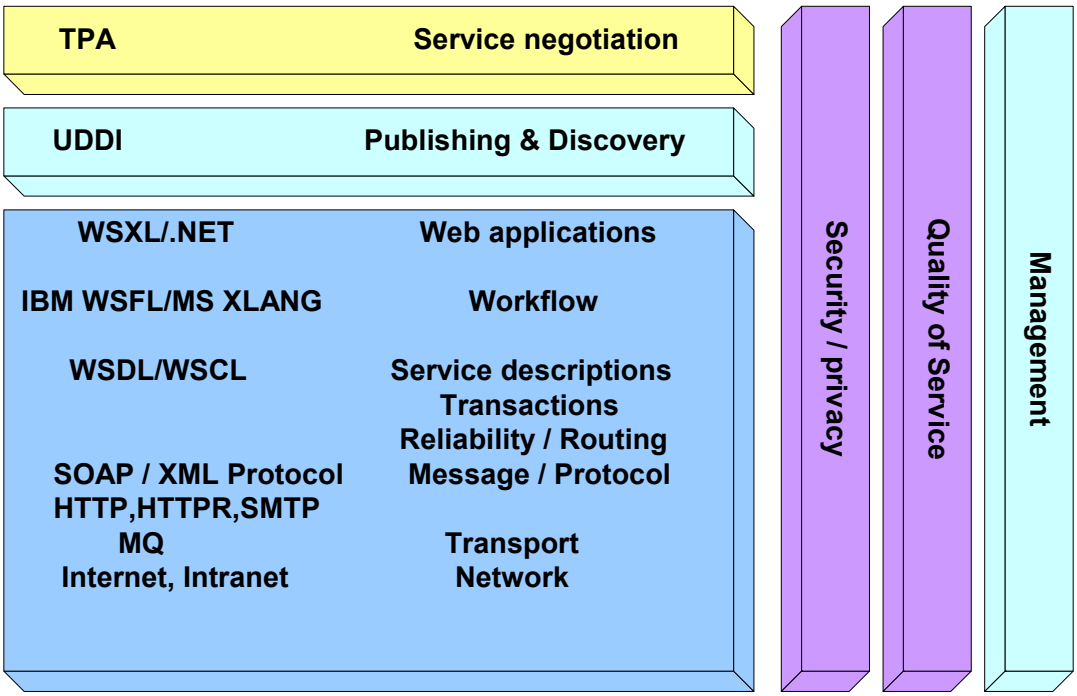

Figure 3. Enabling technologies of web services mapped to the layers of the protocol stack (IBM, 2004).

3.2 Standardization bodies

Several standardization forums are active in the topics related to service architectures. The standardization groups presented below provide an overview of current activities:

- OMG (Object Management Group)[2]: OMG's aim is to set vendor-neutral software standards and enable distributed enterprise interoperability. The most important topics related to pervasive software development are UML, MDA (Model Driven Architecture), MOF (Meta Object Facilities), XMI (Meta Data Interchange) and the telecommunications domain task force.

- OBSAI (Open Base Station Architecture Initiative)[3] and OMA (Open Mobile Alliance) [4] are initiatives for standardization of service architectures for mobile services. OBSAI concerns base stations, and the OMA standardizes application interfaces used in mobile service development. MITA (Mobile Internet Technical Architecture) (Nokia 2002) focuses on the Internet service architecture of mobile terminals.

- IEEE AWG (Institute of Electrical and Electronics Engineers, Architecture Working Group) has standardized architectural descriptions by the Recommended Practice for Architectural Description of Software-Intensive Systems.

- FIPA (The Foundation for Intelligent Physical Agents) has the Architecture Technical Committee that defines architectural specifications providing a service framework necessary to support the end-to-end interoperability of agents.

- ETSI (European Telecommunications Standards Institute) determines standards for developing and testing protocol software.

- ITU (International Telecommunication Unit) has defined the reference architecture model and protocols for computational interactions for Open Distributed Processing.

- IETF (Internet Engineering Task Force) provides specifications for Internet communications.

- 3GPP (Third Generation Partnership Project) has defined 3G mobile system standards.

---

[2] http://www.omg.org
[3] http://www.obsai.org
[4] http://www.oma.org

- Parlay (Parlay Group) and JAIN APIs (Application Programming Interfaces) are specifications for the development of telecom products and services.

- OSGi specifies server functionalities for delivering services between the external Internet and local devices.

- W3C (World Wide Web Consortium) develops interoperable technologies to boost the Web as a forum for information, commerce, communication, and collective understanding.

3.3 Research activities in ubiquitous software development

In the research community, there is an increasing amount of activities in the ubiquitous software area. Research funding organizations are also encouraging topics related to ambience intelligence and pervasive computing. For example, the main objective of the 6th Framework for Information and Software Technologies is to help EU (European Union) citizens' life with smart services available for everyone. The same objective is also seen in the new roadmap for Software Intensive Systems (ITEA (Information Technology for European Advancement) /Eureka), which describes five application domains, namely Home, Cyber enterprise, Nomadic, Infrastructure and basic services and Services and software creation, as being the most important areas in future information technology development.

Although these programs are just beginning, work in this area already started about five years ago. The following organizations have already made successful achievements in ubiquitous computing area, especially related to pervasive software:

- The Fraunhofer Integrated Publication and Information Systems in Germany has worked on support for synchronous collaboration with roomware components. Roomware is a term used to refer to room elements with integrated information technology such as interactive tables, walls and chairs.

- The University of Illinois has developed a middleware infrastructure for active spaces, programmable ubiquitous computing environments in which users interact with several devices and services simultaneously.

- Stanford University has developed a service framework for ubiquitous computing environments that lets users flexibly interact with the services using a variety of modalities and input devices.

- Carnegie Mellon University has developed the Aura architecture for user mobility in ubiquitous computing environments.

- NIST Smart Space Laboratory has developed technologies for wearable pervasive computers. Their focus is on human-systems-interaction.

- VTT Electronics has participated in developing ambient intelligence solutions, mobility across networks and ad-hoc radio systems (VTT Electronics 2004)

Ubiquitous software provides new opportunities for companies that have embedded software in their products and also for pure software companies. Companies in pervasive software markets can be classified into five categories:

- Ubiqiutous infrastructure: telecommunication industry, teleoperators, device and appliance vendors, automation industry.

- Ubiquitous middleware: embedded systems suppliers, middleware software suppliers.

- Pervasive services: software companies in several application domains, e.g. information services, e-commerce, healthcare, entertainment, human-interaction software technologies, and measuring technologies.

- Pervasive development methods and tools: IT (Information Technology) companies specializing in real-time, embedded and communication software technologies.

- Pervasive service providers: distribution and maintenance of pervasive services.

3.4 Software architecture of ubiquitous systems

This section represents the architectural styles and patterns as well as the architectural models and types of components applied in ubiquitous software systems. Architectural styles and patterns are generic software technologies used for specifying reference architecture. OBSAI specify mobile ubiquitous computing infrastructure, and OMA tries to promote an open mobile architecture, the reference architecture for mobile pervasive services (see Figure 2). Reference architecture is an architecture that is explicitly defined, its specifications are freely available and accepted and used on a community-wide basis. Thus, reference architecture is a prerequisite for achieving interoperability between software solutions provided by different software suppliers.

The architectural models that are introduced in this section are examples of micro architectures (i.e. enhanced software technologies, see Figure 2) used in pervasive systems. These micro architectures introduce specific component types that play the key role in

achieving architectural solutions that meet the quality requirements defined for specific pervasive computing environments. The survey is not exhaustive; the research community and technology developers are active in this area and new concepts and technologies are emerging each year.

3.4.1 Architectural styles and patterns

The purpose of using architectural styles and patterns is to assist the architect of a ubiquitous software system in selecting appropriate styles and patterns based on the *quality requirements* set for a new service or a new system. The use of styles and patterns is a means to assure that qualitative properties are, at least, considered in the design phase of a system. Quality attributes, such as quality requirements are considered in architecture design and realized in mechanisms, mostly described as design patterns.

Here, the styles and patterns that are appropriate for wireless services are briefly summarized. Styles and patterns have a specific purpose that they were initially designed for Table 2.

Table 2. Architectural styles and patterns and their purpose

Purpose	Styles/architectural patterns
Distribution	Client-Server style Peer-to-Peer style
Communication	Broker pattern
Decomposition	Tiered style Layered style
Task-orientation	Blackboard style Pipes-and-Filters style
Application-orientation	Model-View-Controller pattern Presentation-Abstraction-Control pattern

The *N-tier C/S (Client/Server)- architecture* means an architectural style in which software functionality is decomposed into tiers that communicate in the client-server fashion. The style is a combination of the Tiered style (Kalaoja et al. 2003) and the C/S style in the runtime structure category. The C/S style decouples client applications from the services they use.

Peer-to-peer (P2P) means network architecture, where information is divided between the participating nodes without centralizing it to one server (Parameswaran et al. 2001). In the P2P model, resources can also be switched between the systems (Gutberlet 2000). Variants of P2P are pure, hybrid and agent-based P2P. Pure P2P is well suited to divid-

ing information between limited number of users. In hybrid P2P, the central server is responsible for maintaining a registry of shared information and responding to queries for that information. In an agent-based P2P architecture, the user communicates with agents that are located inside the wireless device and the agents can work on behalf of the human-users. (Homayounfar 2002)

By using the *Broker* pattern, a wireless application can access distributed services by sending massage calls to the appropriate object (like a network server) through the broker (Buschmann et al. 1996). The Broker architectural pattern is applied to structure distributed wireless systems with decoupled components that interact by remote service invocations.

The *Tiered style* is used to partition a wireless system into logically separated tiers. Each tier has a unique responsibility in the system. A tier is logically separated from other tiers in the system, and is loosely coupled with adjacent tiers.

The *layered* architectural style helps decompose the software into strict ordered horizontal layers where each layer provides its higher-level layer or layers with a cohesive set of services with a public interface (Buschmann et al. 1996), (Clements &Northrop 2002). The Layered style is best suited to a system where the tasks can be divided into application specific and generic tasks.

In *Blackboard,* several specialized subsystems assemble their knowledge to build a possibly partial or approximate solution. The idea behind the Blackboard architecture is a collection of independent programs that work cooperatively on a common data structure. Blackboard is best suited to the systems where scalability is needed in the form of adding consumers of data without changing the procedures and modifiability in the form of changing who produces and consumes which data. (Bass et al. 1998)

The *Pipes-and-Filters* style emphasizes the incremental transformation of data by successive components. The Pipes-and-Filters style intends to view the system as a series of transformations on successive pieces of input data. Data enters the system and flows through the components one at a time until they are assigned to some final destination, output or a data store.

The *Model-View-Controller* (MVC) architectural pattern divides an interactive application into three components. MVC emphasizes modifiability and portability by applying separation of input and output-devices and use of the unit-operation of the part-whole decomposition (Bass et al. 1998).

The *Presentation-Abstraction-Control* architectural pattern (PAC) defines a structure for interactive wireless services in the form of a hierarchy of cooperating agents (Buschmann et al. 1996).

3.4.2 Wireless-specific design patterns

This section presents three wireless-specific patterns: reduced mark-up language, connection-less protocols and multiple presentations.

The problem of presenting complex structured information on a limited device can be solved using a mark-up language such as WML, which requires a small footprint browser. The *reduced mark-up language* addresses the problem of representing fairly complex information on a device with limited capabilities. The limitations that are addressed essentially concern limited display capabilities, limited device resources usable for presentation software, and missing development effort for the implementation of ad-hoc presentation software. Since several recent mobile devices come bundled with WML browsers, it is possible to leverage this built-in capability to implement all the presentation-related features on the client side (Kalaoja et al. 2003).

When a client needs a frequent and low latency notification of events from a server on a wireless network, the use of TCP/IP (Transmission Control Protocol/ Internet Protocol) may not meet the latency requirements. *The connection-less protocols*- solution consists of adopting the UDP/IP (User Datagram Protocol/Internet Protocol) and introducing some packet loss detection and recovery mechanism at a higher level. This pattern has the purpose of limiting the bandwidth occupation of the protocol to obtain low latency.

In a network environment, especially with wireless devices, the characteristics of client applications are often very different. Thus, it is very useful to provide the users with a single point of access to a service and automatically adapt to the client/device features. As the user changes a device, he/she must be offered the same independent of the device capabilities. In general, the same service must be presented through different channels as *multiple presentations*. In practice, each device has different capabilities and the details of the presentation can depend heavily on the device or client application capabilities. This pattern provides a method to automatically detect the type of device/client and switch to the presentation mode that best suits it.

3.4.3 Adaptive resource management

The aim of adaptive resource management is to provide software solutions that assist in resolving adaptability required in pervasive computing environments. Odyssey (Noble et al. 1997) monitors resources such as bandwidth, CPU (Central Processing Unit) cy-

cles, and battery power, and interacts with each application to best exploit them. For example, when connectivity is lost due to a radio shadow, Odyssey detects the change and notifies interested applications. Reaction to the notification depends on the application.

Odyssey is a minimal middleware solution realized as a new VFS (Virtual File System) connected to the NetBSD[5] (Net Berkeley Software Distribution) kernel. Odyssey has two kinds of components; a viceroy that is responsible for the centralized resource management, and wardens that are data-type specific pieces of software that provide the system-level support to clients that is necessary to effectively manage data types (i.e. a warden is required in each client).

Odyssey applies data-centric and event-driven architectural styles. The data-centric style defines the fidelity levels for each data type and factors them into the resource management. Event-driven (or action-centric as defined in the paper) communication is used between middleware and applications for providing applications with control over the selection of fidelity levels supported by the wardens.

In Odyssey, the range of adaptation is defined by two end points; laissez-faire (i.e. no system support required for adaptation) and application-transparency (i.e. the system bears full responsibility for adaptation and resource management). The best solution for a particular ubiquitous system is somewhere between these two extremes, because laissez-faire means that applications by themselves have to take care of adaptation and application-transparent adaptation does not support the diversity of applications. An application launch management concept developed for open source components is another approach to adaptive resource management. The concept is intended for low power handheld devices by preventing end-users from overloading the resources of the device with arbitrary actions. Reliable operation of handheld computing devices with minimal user intervention is one of the requirements the concept has to meet. As users are able to install and run third party software on their new multimedia devices, the resource management problem is becoming a major bottleneck. No matter what they install, they still expect perfect operation when considering the rest of the system. The concept extends the X-Window system with the application launch management features and protects necessary resource reservations for critical user space applications. It also includes a concept for generic resource management control and monitoring. (Hongisto 2003)

[5] http://www.netbsd.org/

3.4.4 Proactive service discovery

The contribution of the Proactive Discovery Service (PDS) (Bustamante et al. 2002) is that it allows clients dynamically tune the detail and granularity of the notifications about changes they are interested in. Thus, it provides a mechanim for achieving a scalable service discovery service. PDS is message-based, contrary to W3C's service discovery that is heavily based on RPC (Remote Procedure Call), although the necessity of message-based interaction model is already identified (Vinoski 2002).

PDS extends an ordinary directory service in three ways:

1) It links a channel and a change notification with each object managed by the directory service.

2) It customizes notification channels thorough client-specific filters.

3) It defines a leasing model for client registration to a notification channel in order to simplify the handling of client failures.

The advantage of the proactivity model is that it allows clients to transform from controlling pull-based passive interfaces to trade control for performance because message traffic is generated only when updates occur. The PDS architecture includes three main components: PDS clients, servers and object owners. Clients discover available objects in the environment and become aware of any change in them that could affect their functionality and/or performance. Object owners publish their objects through the directory service. Servers act as mediators. In PDS, related information is organized into well-defined collections called entities, where each entity represents an instance of an actual object type in the environment. Each object has an associated set of properties (or attributes) with particular values. Entities may be bound to names in different contexts and each context contains a list of name-to-entity bindings. In turn, contexts may be bound to names in other contexts, building an arbitrary directed naming graph.

Scalability of the global naming space is obtained by dividing it into sub-spaces and assigning these sub-spaces to domains, each with a single root context. Security mechanisms are not available in PDS, and the extensions of the language for customization are also missing. PDS is a general directory service but shares a number of architectural ideas with GMA (Grid Monitoring Architecture) (Tierney et al. 2002). The authors of PDS mentioned further exploration of PDS with AIMS (Adaptive Introspective Management System) and use of proactivity to enhance the robustness of widely distributed services through flexible replication strategies, dynamically adaptive server hierarchy management and automatic failure recovery. Dynamic service discovery with self-adaptation is also considered a promising approach to a generic service discovery service in which all kinds of services can interoperate with each other (Vinoski 2003). This

idea is based on Apple's Rendezvous, which focuses on easily connecting computers and devices through a multicast interaction model and trader-like mechanisms. However, Rendezvous applies to software as well, and Apple uses it for its iChat instant messenger system and file sharing.

3.4.5 Context-aware coordination

The MARS (Mobile Agent Reactive Spaces) architecture (Cabri et al. 2002) introduces the coordination medium (CM), a service that is typically associated with an Internet node or with a local administrative domain of nodes, in charge of acting as a mediator for all coordination activities in that site (i.e. node or domain). CM provides, through a specific API, the capability to both access the local resources of a site (i.e. agent-environment coordination) and to interact with other local agents and other application agents (i.e. inter-agent coordination). Any coordination model, such as meetings, event-channels and tuple spaces, can be applied to providing the API for agents. The applied tuple spaces model can easily be integrated with the current web scenario.

The MARS architecture applies many styles and patterns. The intra-agent architecture depends on the task to be performed, but the main style is the independent-components style. Interactions between agents and agents and the environment define the main styles to be applied; the peer-to-peer style for networking and the rule-based style for defining global rules and environment-specific coordination laws. Moreover, MARS reactions can be combined in a pipeline and therefore, a hierarchical Pipes-and-Filters style can be applied to install and uninstall reactions on a tuple space. The code and behavior of a reaction can also be changed without changing the agent and/or the other reactions because the agents and reactions are implemented separately, which leads to advantages in code development and maintenance.

An adapter with the adapter API running on top of middleware is a solution for runtime binding of components (Chiang 2003). This binding model is based on the two types of interactions:

1) adapters communicate with each other through a mediator or a facilitator or

2) adapters communicate with each other directly.

The first approach decreases implementation complexity at the expense of increased interaction overhead. Component interfaces are defined by the server names, operation names, number of parameters in the interface, parameters' orders, types of parameters, sizes of parameters and directions of parameters such as IN, OUT, or INOUT. Adapters invoke components based on a component's name and the operation name. The inter-

face needs to be agreed on by communicating components. It happens by registering interfaces to the associated adapters of these components. The biggest drawback of the approach could be overhead, which can make this concept unapplicable in real-time ubiquitous systems and some mobile ubiquitous systems.

3.4.6 Multi-agents

A G-net system (Xu & Shatz 2003) includes a number of G-nets, each of them representing a self-contained module or an object. A G-net is composed of two parts: a special place called Generic Switch Place (GSP) and an Internal Structure (IS). GSP provides the abstraction of the module, and serves as the only interface between the G-net and other modules. IS, a modified Petri net, represents the design of the module. The need for extended G-nets is justified as follows:

- Multi-agent systems are developed independently by different vendors and agents may be distributed across large-scale networks. Therefore, agents require a common communication language and common protocols.

- The agent communication model is usually asynchronous and an agent may decide whether to perform actions requested by some other agents. However, the standard G-net does not directly support asynchronous message passing and decision-making.

- Agents are designed to determine their behavior based on individual goals, their knowledge and the environment. They may autonomously and spontaneously initiate internal or external behavior at any time. The standard G-net models can only directly support a predefined flow of control.

The Planner module is the heart of an agent; it can ignore an incoming message, start a new conversation or continue with the current conversation. Through the Planner module, the Goal, Plan and Knowledge-base modules of an agent are updated after the processing of each communicative act that defines the type and content of a message, or if the environment changes. The Planner module is both goal-driven and event-driven because the transition sensor may fire when any committed plan is ready to be achieved or any new event happens. The Planner is also message-triggered because certain actions may initiate whenever a message arrives. A message is represented as a message token with a tag of internal/external/private.

IS consists of incoming message, outgoing message and private utility (that can be called by the agent itself). The incoming/outgoing message section defines a set of Message Processing Units (MPU), which corresponds to a subset of communicative acts. A new mechanism called Message-passing Switch Place (MSP) is introduced for asyn-

chronous message passing. When a token reaches an MSP, the token is removed and deposited into the GSP of the called agent and the calling agent continues to execute its next step.

A mix of several styles is applied in agent-oriented software development. IS is similar to the Pipes-and-Filters style. The Planner acts as a broker, including independent modules, and the dispatcher pattern has been used several times. The major benefit of the multi-agent architecture is the amount of hot-spots to be applied to extensions, modifications and reusability. However, applying it requires a middleware solution upon which agents can be executed, and in practical solutions it may also be too laborious.

3.4.7 Models for heterogeneous environments

3.4.7.1 Meta models

Gaia (Roman et al. 2002) exports a service to query access and use existing resources and context, and provides a framework to develop user-centric, resource-aware, multi-device, context-sensitive, and mobile applications. Gaia's main contribution is the functionality it provides as the result of the interaction of individual services. This interaction provides users and developers with an abstract ubiquitous computing environment as a single reactive and programmable entity instead of a collection of heterogeneous individual devices. Gaia has been applied in a ubiquitous computing environment in a very similar way to digitally augmented meeting rooms.

Gaia provides five basic services based on top of CORBA middleware: event manager, context service, presence service, space repository and context file system. For example, the event manager uses the CORBA event service as a default event factory. Although similar kinds of names as patterns are used, Gaia does not have a clear connection to any style, i.e. it is not a software engineering approach but gets its origin from the research of augmented reality. However, the Gaia system is a mix of the event-driven style, independent-components style and rule-based style. The proxy pattern is used on behalf of the physical entities (i.e. application, service, device and person) in the physical-entity presence subsystem. Periodically, notification is sent as heartbeats of the present services and applications and this means the use of the observer pattern. Space repository stores information about all software and hardware entities in the space as XML descriptions including the properties of services. The repository is based on the CORBA Trader service. The context file system (CFS) uses application-dependent properties and environmental context information to simplify many of the tasks that are traditionally performed manually or require additional programming. CFS is composed of mount and file servers. One mount server maintains an active space's namespace. For

the context model, Gaia uses first-order logic and Boolean algebra, which allow easy writing of rules to describe context information.

In Gaia, applications are partitioned among a group of coordinated devices. The application framework consists of a distributed component-based infrastructure (derived and refined from the MVC pattern), a mapping mechanism for customization and a group of policies that defines sets of rules for customization (concerning instantiation, mobility, reliability and composition). The mapping mechanism of MVC defines two application description files: an application generic description and application customized description (ACD) that is an active-space-independent description. ACD defines how the component is assembled, i.e. a list of components and how they are allocated and initialized.

Gaia differs from Aura in that Gaia emphasizes space programmability and users ability to configure their applications to benefit from the resources in their current space.

3.4.7.2 Component types

The main goal of the Aura architecture (Sousa & Garlan 2002) is to maximize the use of available resources and to minimize user distraction and drains on user attention. The model is simple, including only four component types:

- The Task Manager embodies the concept of personal Aura.
- The Context Observer provides information on the physical context and reports relevant events in the physical context back to the Task Manager.
- The Environment Manager embodies the gateway to the environment.
- The Suppliers provide the abstract services from which tasks are composed.

Each environment has one instance of the Task Manager, Context Observer and Environment Manager that cooperate with the corresponding components in other environment.

The Task Manager strives to minimize user distraction in the face of the following changes:

1) The user moves to another environment (i.e. migration of personal information).
2) The environment changes (monitoring of QoS (Quality of Service) information).
3) The task changes (i.e. indicators from the user).
4) The context changes (monitoring context-dependent constraints).

For task migration, the service status is provided as a markup representation. Suppliers of a given service type share a vocabulary of tags and the corresponding interpretation. Existing applications can also be wrapped to the Aura architecture.

Context Observers in each environment may have different degrees of sophistication, depending on the sensors deployed in that environment. Examples of sophistication dimensions are user recognition, location, activity and other people in the vicinity.

When Suppliers are installed in an environment, they are registered with the local Environment Manager. The registry is the base for matching requests for services and it also keeps a record of available capacity.

Aura supports dynamic reconfiguration in a transparent way and hides the variation of low-level interaction mechanisms from one environment to the next, implemented by connectors. Thus, deployment of suppliers differs across devices and the deployment may change dynamically. The interaction mechanism in Aura may be CORBA, while in another it may be COM or simply RPC. Standard interfaces of components are defined as ports.

Although (Sousa & Garlan 2002) does not mention the styles and patterns used in Aura, the main architectural style seems to be independent components style, using at least the observer and bridge patterns. However, self-awareness and adaptability of the environment is addressed at two levels. The infrastructure level monitors the availability and performance of components and communication (i.e. coarse-grained adaptation). At the lower level, system components themselves are endowed with the ability to adjust their operation following the variation of available resources.

3.4.7.3 Generative model

The architecture in (Ponnekanti et al. 2001) includes the Interface Manager and one or more generators defined in the Generator Database. While generating code, the selected generator uses information about the workspace context from the Context Memory. Code generation requires information about services, appliance and workspace. It is produced in the following way: services send a beacon about their presence including a service description, an appliance supplies an appliance description, and workspace context is stored in a central datastore called the context memory.

The generative model uses proxies that are applied to achieve flexibility for different appliances. The main architecture is an event-based, blackboard and rule-based system that uses thoroughly defined service and device descriptions as input for code genera-

tion that is integrated as a part of supporting services. Thus, the framework also supports evolvability. Design patterns are also utilized in code generation. Although the current realization is event-driven, the communication can be changed to RPC/RMI (RPC/ Remote Method Invocation) or message-based communication.

This section surveyed some advanced and enabling software technologies, i.e. architectural styles and patterns as well as some design patterns applicable for ubiquitous systems. The aim is not to provide a technology roadmap but to introduce some existing technologies and promising approaches already applied to the ubiquitous systems in a specific context. Although there are several technologies that have already proven their applicability in pervasive computing, there is also a lot of work to do, especially in defining micro- and macroarchitectures specific for ubiquitous systems and promoting standardization of these architectures as reference architectures and open standards.

3.5 Development aspects of ubiquitous software

This section explores how the characteristics of ubiquitous software affect the software development and methods used in the development.

3.5.1 Towards service-oriented software development

Service-orientation is a new paradigm just entering the embedded software field. By service-orientation software engineering it is meant software architectures and development practices needed in the development of software services for networked embedded systems. Service architectures and alliances such as OBSAI and OMA provide standards, which facilitate realization of service-orientation in ubiquitous software development as well. A software service can be defined as a self-contained Internet based application, capable of completing tasks on its own and able to discover other services and use them to compose a new higher level functionality. A service is easily found, used when needed and then discarded.

Software organizations that exploit service-oriented software development are rather small and in a continual process of change. These companies act as service developers, content providers or/and service providers. Companies that produce appliances, devices and/or systems have different roles. They act as integrators and aggregators by producing the infrastructure through which the services are offered to the mass markets. A company may play several roles at the same time, and in the best case, all these roles are represented by various companies in a global networked coalition. (Zhou & Niemelä 2004a; Kallio et al. 2004; Niemelä et al. 2003)

In service-orientation, software developers prefer exploitating the components available in the marketplace to produce the most effective software in a short time rather than programming from scratch. In service-orientation, software is produced as a particular service that conforms to a service standard technology. A system may be composed, executed, maintained and evaluated in the way of online procuring, engaging, and changing. (Zhou & Niemelä 2004a)

The differences between application-oriented and service-oriented software development are shown in Table 3 (Zhou &Niemelä 2004a).

Table 3. Comparison of application and service orientated software engineering.

Application-oriented software development	Service-oriented software development
• Supply-side method • Product • System • Ownership • Rigid-boundaries • Technology first • Large-scale organization • Several months	• Demand-side method • Instant service • Particular software when needed • Loose-coupled • Unfixed boundaries • Non-technology first • Small-size organization • Procurement first

Suppliers that typically dominate application-oriented software development are closely coupled with customers' business problems and software solutions. Suppliers offer products and systems, and customers buy and own the applications. The supply-side methods, driven by technology advance, have worked well for systems with rigid boundaries of concern such as traditional embedded systems. Software development may also benefit from large-size and product-focused organizations. However, a slower time to market and high cost is associated with the maintenance and evolution of the application. (Zhou & Niemelä 2004a)

In service-oriented software development dominated by customers, an application is broken down into finer grained parts. Organizations are small in size and focus on software market and maintenance. Customers have no interest in owning the application, but use software as needed. This means that functionality is delivered as a service. Each time the functionality is required, service elements are identified, executed and discarded (i.e. an instant service). Non-technology issues, such as supply contracts, terms and conditions, drive the demand-side methods. (Zhou & Niemelä 2004a)

Service-oriented software development fits well to the ubiquitous software development because end-users satisfaction is the key driver in pervasive computing. Application of

service-oriented software engineering, however, requires further investigations, at least, in the following issues (Zhou & Niemelä 2004a):

- **Service evaluation.** Customers should be allowed to 'take-try-and–use' a service in a cheap or cost-free way when needed.
- **A unified service middleware standard**. A unified service middleware standard would assist in service description, requisition expression, service composition, implementation and combination that would be supported by service providers, brokers and customers.
- **Goal-driven requirements engineering**. Services are targeted to a potential customer group rather than an individual customer. That is why organizations face implicit requirements, from which explicit functional and quality requirements have to be derived in order for a customer to be able to select various options for his/her needs.
- **Value-adding to legacy software**. By building the legacy code as a service, extending the service and promoting the service quality will lengthen the service life cycle.

3.5.2 Adding quality to legacy software

Ubiquitous computing means the use of in-house and 3rd party components that also have to meet the specific properties of ubiquitous systems such as security and reliability. All programmers do not have the required special skills and know-how to know how these qualities can be achieved, and that is why these properties are often ignored. One solution is to add quality properties into components by post hoc, which means that a component is transparently translated into a component that takes the specified quality properties into account. The solution may not provide the properties completely, but the approach improves the quality of software drastically. (Nakijima et al. 2002)

One approach for adding quality properties is Aspect-Oriented Programming (AOP). In AOP, quality properties are defined as aspects which are merged into base components. In the most favourable case, this can be achieved by recompiling a component using an AOP-aware compiler without substantial modifications to the source code. This approach enables the components to be adapted according to the characteristics of underlying platforms. There are also proposals to translate Java binary codes for adding quality properties by post hoc. However, there are several issues to be considered. Firstly, how the QoS evolution can be achieved transparently from a client program. If adding quality features changes some assumptions of the component, the correctness of the client application may be violated. Thus, rigorous API semantics are needed for defining (middleware) components, and the assumptions need to be checked after adding the

new features. Secondly, the current AOP techniques require an understanding of the internal structure of the base component in order to be able to define aspects. However, it is neither reasonable nor cost effective to learn the internal complexity of a large component such as Linux, Java or CORBA. Thus, it is important to export a high level description of the component structure for defining new quality aspects. (Nakijima et al. 2002)

Another approach to add quality properties transparently is a resource kernel. The resource kernel monitors the resource utilization of all applications, and enforces their behaviours if they violate the QoS requirements specified by them. By adding the resource kernel to a non real-time operating system it is possible, for example, to convert a traditional operating system into a real-time operating system. The resource kernel provides several primitives that control resources explicitly and an application should be modified to invoke these primitives, so that the application would behave predictably. The primitives are meta-interfaces that control internal resources such as CPU, memory and network bandwidth and support portability of the application. (Nakijima et al. 2002)

In ubiquitous computing, a variety of platforms are needed and they have to be able to work together. If a commercial middleware is used, it is important to consider how to exploit advanced characteristics provided by the underlying platform. One approach is to add meta-level interfaces or QoS parameters to control the internal algorithms of the platform components. There is a conflict because a generic interface usually hides low-level characteristics of the underlying platforms. (Nakijima et al. 2002)

In the software development, trade-offs among different qualities have to be taken into account. For example, a programmer needs to consider non-functional properties such as timeliness, precision, accuracy and consistency to build a service or an application. Because it is impossible to satisfy all requirements, a programmer must consider which requirements he/she has to focus on as the decision affects the program's architecture. For example, distributed applications should take into account three properties that affect the quality of a system: consistency, availability and network partition. Building portable software requires making assumptions explicit, because these assumptions are necessary to ensure the correctness of a program. (Nakijima et al. 2002)

3.5.3 Agile methods in ubiquitous software development

The traditional software development consists of a series of processes and rules, which make software development planned and controlled. In the development of pervasive software, there are more unknown requirements, more customers' expectation and more changes during the software development, all of which call for a more adaptive software development, i.e. agile methods. Agile methods balance the non-process hacker model new change driven development model in order to obtain a relatively satisfactory out-

come, and therefore applying them in the development of ubiquitous applications is considered a promising approach.

The core set of agile methods (Abrahamsson et al. 2003; Zhou & Niemelä 2004b) includes eXtreme Programming (XP)[6], Feature-Driven Development (FDD)[7], Adaptive Software Development (ASD)[8] and Scrum[9].

- XP preaches the values of community, simplicity, feedback, and courage. XP is aimed at small and medium-sized teams. Communication and coordination between XP members should be enabled at all times. The common XP culture and advanced technology shall guarantee the XP project's successful implementation.

- FDD is a process-oriented, model-driven, short-iteration process. One iteration should take from a few days to a maximum of two weeks. FDD is suitable for above medium-sized teams with rich modeling experience, new projects starting out, projects enhancing and upgrading existing code, and projects tasked with the creation of a second version of an existing application.

- ASD replaces the static plan-design-build life cycle with the dynamic speculate-collaborate-learn life cycle. ASD does not have built-in limitation for its project size and field. An ASD project can be a cross-domain complex system, and ASD needs techniques for enhancing inter-team collaboration to support distributed development.

- Scrum is an empirical project management approach applying the ideas of industrial process control theory to systems development resulting in an approach that reintroduces the ideas of flexibility, adaptability and productivity. Traditional processes are designed only to respond to the unpredictability of the external and development environments. The Scrum process is quite flexible and provides collaboration, training and learning mechanisms for developers to share tacit knowledge and devise the most ingenious solutions. Scrum is suitable for small teams of less than ten individuals.

Present agile methods can be characterized by the project-driven, intra-enterprise development, object-oriented modeling and people-centric development. However, agile methods ignore the role of software architecture and due to the iterative nature, they easily lead to design erosion. Agile methods also require improvements in scaling them to large development teams, integration and enterprise across application development.

---

[6] http://www.extremenprogramming.org/
[7] http://nebulon.com/fdd/
[8] http://www.adaptivesd.com/
[9] http://ww.controlchaos.com

In summary, the issues to be further investigated and applied to ubiquitous (application) development on agile methods include (Zhou & Niemelä 2004b):

Process. Light-weight methods improve the plan-driven waterfall process. However, it would be more beneficial for software organizations to adopt the Win-Win alliance strategy to respond to market changes. Because the increasingly unpredictable software market forces software organizations to pay more attention to finding business opportunities, it is necessary for the methods to combine the processes of component procurement, organization alliance and software knowledge discovery into software development.

Modeling. Present agile methods are all based on object-oriented methods. The object-orientation is suitable for modeling intra-enterprise business. However, today's projects are generally cross-domains and cross-enterprises, and therefore, it is necessary to study the domain-oriented and pervasive computing-oriented modeling methods. In recent years, the ontology-oriented method has been used for modeling enterprises, domain knowledge and natural language. The ontology-oriented software model will facilitate the communication and inter-operation among people, enterprise, computer and software.

Software design rationale capturing. Design rationale is the underlying intent and logical information behind design decisions. The few existing systems that can capture the rationale are severely limited. The first software version should be done in a short iteration cycle (e.g. two weeks), which expects more maintainable software. Capturing software design rationale will enhance software maintenance and reengineering.

Human capital. All present agile methods think that the human factor is the most important factor (e.g. technology, process) impacting software development, but they do not point out an effective method of managing people. In fact, managing human capital means managing and developing the tacit knowledge hidden in people's heads. Knowledge is created through the interaction between tacit and explicit knowledge. Four different modes of knowledge conversion are: socialization (from tacit to tacit); externalization (from tacit to explicit); combination (from explicit to explicit) and internalization (from explicit to tacit). Obviously, managing the people factor is the process of computerizing the tacit knowledge. It is necessary for agile methods to study how to manage and develop human capital in software development. (Zhou & Niemelä 2004b; Kähkönen & Abrahamsson 2003)

3.5.4 Software testing

Testing is an integral part of the software development process. When software is tested, it is executed with the intention of finding errors, and to determine that the software meets its requirements (Myers 1979; Latvakoski 1997). Testing validates the com-

pliance of a system to functional and non-functional requirements (e.g., performance). Typically, 30-50% of system development effort is used for testing. In ubiquitous computing software, the importance of testing is estimated to be increasing due to the requirement of interoperability.

The computer-aided software testing (CAST) is one answer to lowering the cost of testing. This refers to methods and tools supporting automated or, at least, semi-automated testing. Test case generation, test execution and test result analysis are much easier for a human tester when these activities are supported by advanced CAST tools. However, an essential requirement is that the methods and tools can be seamlessly integrated together with the test execution platforms both in simulation, integration and system testing.

For the time being, testing is done relying on traditional testing practices, ad-hoc approaches, and proprietary solutions. Ubiquitous computing requires defining new testing methods for the special requirements that ubiquitous computing presents: new testing processes and testing environments that enable testing of ubiquitous computing software in real environments. The challenge is to represent the non-existent software modules and network elements as well as make it possible to test in workstation environment. This refers to the need for *test software*, which is usually required to enable test execution (Latvakoski 1997). The final phases of testing, i.e. system and acceptance testing, require a real execution environment. The challenge is to provide a real execution environment for ubiquitous computing software, which is not a trivial environment to setup and manage but includes several devices and network elements with different interfaces.

One approach to automated testing environments is to use tree and tabular combined notation (TTCN), which is a language for test specification and implementation. The latest version of the language (TTCN-3) should be applicable in many application areas, not just conformance testing. Typical areas of application are protocol testing (including mobile and Internet protocols), service testing (including supplementary services), module testing, testing of e.g. CORBA based platforms, APIs etc. TTCN-3 can also be used in many other kinds of testing, including interoperability, robustness, regression, system and integration testing. TTCN-3 is an abstract language that is adaptable for different kinds of execution environments and system under tests. It also provides a mechanism to distribute testing over several nodes, a means for automatic test execution, and interfaces to define different time handling mechanisms from simulated time to real time.

In service-oriented software engineering, testing techniques such as TTCN-3 are applied to testing infrastructure and middleware services, but they may be too expensive and time-consuming for end-user service development. Software services are fine-grained software pieces developed for a particular environment and validated by using them as a

composition of end-user services in real environment. This means that the testing platform/component should be part of a service package and easily used by the service users as defined in the specifications and specific test cases.

3.6 Summary

Enabling software technologies, i.e. standards, reference architectures and generic modeling and component technologies provide the basis for the development of ubiquitous software. Advanced software technologies enhance these technologies by providing concepts, micro-architectures and mechanisms to implement required supporting services for the special requirements of ubiquitous software. Meta-models, component types and generative models support heterogeneity of software technologies. Meta-models also provide a bridge between various component models and hereby support interoperability of software systems. Context-aware coordination and multi-agents architectures assist in achieving mobility of services. However, there are still lack of supporting services and technologies needed for full mobility as well as achieving security, privacy and survivability in mobile applications.

Adaptive resource management is a concept for achieving adaptability for changing network and computing resources. Although context-aware co-ordination is one of the first steps towards self-organisation, it still keeps us waiting on better and more efficient technologies. There are several new concepts and ongoing research activities related to augmented reality and content scalability. However, most of their achievements are not applicable to practical ubiquitous systems, yet.

Software development methodologies are evolving fast. Contradictory goals to develop software faster, better and cheaper bring software engineers to face an extremely difficult situation. This leads to use as much as possible existing software based on standard information technologies, usage of agile methods in the application development and breaking down the infrastructure and middleware solutions to services with maximum independence. Thereafter, virtual and distributed organisations develop services that are integrated by the service providers. In the extreme case, while an end user defines the features he/she wants from a service, the feature selection activates self-organisation mechanisms that assist automatic service creation.

4. State of the practice in Finnish R & D

For obtaining views about the state-of-the-practice and future of ubiquitous software representatives of eight companies were interviewed and a questionnaire was sent to 80 companies. The aim of the interviews and questionnaire was to obtain views of the companies about current state and future challenges of ubiquitous software business.

The companies presented in this research function in the following roles:

- device vendor for consumer electronics to mass-markets (one company),
- teleoperator (one company),
- system deliverer of industry (one company),
- software company (six companies),
- component- and service provider for industry and commerce (two companies),
- content provider (press and broadcasting, one company),
- component-provider for automation industry (one company),
- telecommunication company (two companies), and
- others (one company).

The number of companies in brackets indicates the number of companies functioning in the role in question.

Based on the results received from the interviews, a questionnaire was made and sent to 80 companies. Replies to the questionnaire it was received from 12 company representatives. In the interviews and the questionnaire, the representatives of the companies were asked their views about the following topics: problems and challenges in software development, product development and business environment.

The persons were also asked to propose useful research topics and ways to transfer the knowledge into industry. The following sections present a summary and analysis of the information received in the interviews and the responses to the questionnaire divided into different aspects of software development.

The following section present background knowledge of the researched companies about ubiquitous software, their points of view in ubiquitous system and software development, architectural design and analysis, business. Domain-specific aspects of the researched companies are also summarized in the following section.

4.1 Background knowledge and current state of ubiquitous arena

The persons involved in this research had various definitions for ubiquitous computing and background knowledge about the topic. The definitions given for ubiquitous area are viewed in the following from several perspectives:

General view:

"Ubiquitous computing combines software and hardware."
"We do not use the concept "ubiquitous computing" anymore, but have focused on specific areas of ubiquitous computing."

Service view:

Ubiquitous service "means everywhere, as easy to use a service as possible". "Ubiquitous applications are normal mobile applications that have various parameters and stimuli from the physical world (place, context etc.)."
"Ubiquitous applications are hidden in the environment."

Technological view:

"Ubicom is a short-range network- several devices that contact each other via a network."
"Ubiquitous devices are devices that control events of the physical world and connect several devices and their events, are able to learn from the routines of the user."

As the above definitions prove the concept of ubiquitous computing/software varies in different companies and domains. Different persons regarded ubiquitous software as an interesting topic although their background knowledge about the topic varied very much. The persons had three different views about the current technological level of ubiquitous computing:

1. The technology for ubiquitous applications exists.
2. The technology for ubiquitous applications exists partly.
3. The technology for ubiquitous applications does not exist.

The difference of the above views is partly due to varying definitions and knowledge about ubiquitous computing and software. The representatives of the companies saw that the portion of software in systems will increase in the future, although separation of hardware and software was regarded as problematic. The ubiquitous area was seen to have several open questions that need to be solved in the future.

4.2 System development

The representatives of the companies regarded achieving standard interfaces and interoperability between applications and devices as the main challenges to development of ubiquitous software. Interoperability is seen as a basic factor for making ubiquitous computing and software successful. At the moment, devices of different manufacturers and applications do not interoperate, as standards are too loose. Some people think that achieving a unique standard is impossible due to tough competition. Interface standardization is regarded as another big challenge to ubiquitous software development.

The following were seen as other technological challenges to ubiquitous systems:

- security (privacy, customer identification, management of user rights),
- power consumption; the challenge of ubiquitous computing is to develop low-power devices for different places of use,
- openness,
- scalability,
- management of the decentralization and the huge wholeness,
- invisible data transfer infrastructure to different places cost-effectively,
- dynamic networking, ad-hoc communication and context recognition,
- ability of the system to filter information,
- automatic error handling,
- creating small and effective embedded systems,
- modifying products after their implementation,
- focusing on certain technologies, and
- the ability to draw conclusions, and implementation of a metadata-layer.

Many of the above challenges (like scalability and decentralization) are caused by globalization that forces the systems to expand and be usable from different locations via different devices. Another challenge to system development in ubiquitous software is increasing reusability so that components can be reused in same company/other companies.

According to the persons interviewed, the current standards created by different groupings pursuing their own interests were weakly usable in embedded systems and too loose, therefore causing interoperability problems. Proper architecture requires definition of interoperability, interfaces and functions. The following are regarded as challenges to the standardization of ubiquitous:

- achieving open, standard interfaces for different devices and for a limited environment, and
- standardization of software platforms and data received from different sources.

4.3 Architectural design and analysis

The representatives of the companies consider currently available architectural design and evaluation methods to be inadequate, different components and interfaces variable, and standards too loose to achieve interoperability.

The following were regarded as challenges to architectural design:

- design of standard application interfaces,
- developing architectural description methods that can be understood by everybody,
- increasing component-based architecture- thinking,
- developing a common component-library to product-lines and common middleware for all platforms,
- getting general middleware to support the architectural design of ubiquitous applications,
- interoperability of architectures, and
- gaining more knowledge about architectural and software design.

Interoperability also presents the biggest challenges to architectural design of ubiquitous systems. Efforts like OMA and OSA aim to achieve interoperability of interfaces and applications, but there is still a lot of work to do in order to gain fully functional interoperability.

4.4 Software development

The software development of ubiquitous applications will - according to the company representatives - face several challenges:

- achieving a ubiquitous way of thinking,
- increasing peoples' competence and trust in others' actions/software,
- sharing programming experiences,
- finding suitable development tools for developing ubiquitous applications,
- achieving compatibility of the versions between different manufacturers,
- creating easy-to-use applications,

- increased complexity of programs,
- successful combination of software and hardware, and
- development in an environment with scarce resources (limited memory, supply of electricity).

The problems of software development were seen to be difficulty in reusing current software and making it reusable, short-time-to-market, high product development costs (especially for small companies), inadequate requirement specifications and documentation, knowledge about software design and poor quality of specifications and code due to a lack of time.

Changing the way of thinking towards openess and wider cooperation is one of the main challenges according to the interviewed persons. Another challenge is to make easy-to-use applications fast as programs become even more complicated and the amount of available resources is limited. The proportion of software is estimated to increase in systems in future.

From the applications point of view, the following are regarded as future challenges:

- the small size of devices/ applications that also limits other features,
- presenting the semantics of information,
- getting rid of the errors and resetting needs of the devices,
- taking into account all requirements,
- developing applications that are easy to use, personalize and reliable,
- handling the increased complexity, and
- personalization.

Especially in the case of mobile and wireless devices, people are accustomed to resetting devices and this need should be removed in future. Companies functioning as subcontractors regard taking all requirements into account difficult due to inadequate requirements specifications.

At the moment software testing was seen to make up too large a portion of the software development costs, and decentralization increases the interoperability problems even more by increasing the costs of testing. Added testing is not seen to increase quality, but the procust should be qualitative right at the beginning.

4.5 Business challenges to ubiquitous computing

The value chain of ubiquitous computing is regarded as unclear and therefore answers should be defined to the following questions:

- Who pays for building the system and whose is it (consumer, teleoperator, ...)?
- Who owns which part of the service and who gains profit from it (costs vs. profit)?
- What is the level of service the end-user receives and is it adequate?
- What is the role of the operator?
- What kind of new concepts and visions could exist (where to achieve cost savings)?
- Who pays for what and is the customer willing to pay?
- What is the revenue logic of the P2P approach?
- Who is the provider of the service / who does business?

Concerning the customer service of ubiquitous services, definition of who is responsible for the functionality of the service to the customer and who fixes the bugs or whether the system could automatically fix the problems is required. The infrastructural challenge of ubiquitous computing is to define who pays for building the infrastructure and whose is it.

According to the interviews, the revenue logic of ubiquitous services is seen as unclear although the area has revenue potential and competent people. The services should be almost free to the consumer and money could be collected via advertisements as everything costs either directly or indirectly (there is no such thing as a free lunch).

The following topics were seen as business opportunities:

- security (as people are willing to pay for this),
- business applications,
- software developed from the initiative of device manufacturers,
- local advertisement,
- services that help daily and group communication,
- home environment,
- industry automation, and
- consumer electronics.

4.6 Special ubisoft- features of different domains

The following Table 4 illustrates domain-specific issues of ubiquitous software according to the role of a company.

Table 4. Domain-specific issues of ubiquitous software.

General comments	Challenges	Open questions	Business potential
Device vendor for consumer electronics to mass-markets			
• Current systems are closed and locally defined	• Software should be adaptable so that data could be picked up from different sources. • It should be possible to transfer data between databases • Data should be filtered to avoid an excessive amount of information • The system should be open and know what information can be given to which device	• How does the system collect information? • How much data can be transferred and where so that the system does not crash? • What information can be received and used and where does it go? • Who is responsible for functionality of the service towards the customer? • How are the systems connected to each other?	• Advertisement • Security
Teleoperator			
• The device field will be mixed (pocketPCs, SmartPhones) • Legislation prevents the teleoperator from managing the whole value chain, and therefore the value chain should be split between several parties	• Providing side-functionalities for maintaining the infrastructure • Achieving global roaming that is perhaps technically possible but not yet politically possible	• What is the role of teleoperator in the ubiquitous world?	
System deliverer for industry			
• There are no wireless sensors for implementing ubiquitous systems • Expansion of sensor networking would have a large effect on their product development. • The traditional software development process does not work	• Implementing a sensor to various places (surface of paper)	• Where are the observations saved and how they can be found? • Is there enough memory for saving the observations?	• Diagnostics (like temperatures, vibrations,...) • If sensors could be implemented more locally, the need for travel would decrease thanks to the possibility of remote monitoring

Continues...

Software house			
• Power consumption is too high at the moment	• Making the infrastructure invisible • Managing the whole value chain • Defining standard interfaces	• Who owns which part of the service?	• Business applications
Component and service provider for industry and commerce			
	• Intelligent field-machines should be based more on communication techniques (like IP) • Good remote-update methods to networked field machines should be developed • Standardization of software platforms • Running Linux with low power requirements	• Who pays for the infrastructure investment?	• Providing remote services
Content provider (press and broadcasting)			
• At the moment, home networks are missing essential parts • Everything will become IP-based and WiFi-networks will become more general • Wires have still not disappeared	• Context has no importance to them as their content is consumed anywhere	• How is the interface of media content and ubicomp and where is the business? • What new media services do home networks bring? • How big are the markets for tailored media?	• Wireless local area networks that would become general • Tailored context-aware media and • Home applications
Component-provider of automation industry			
• Software proportion in their product is 25 % and rising	• Intelligence of the motor and achieving communication between machines • Ethernet applied to industry-conditions		• Wireless control and surveillance of house aspirators • Proactive maintenance • Easy-to-use devices that guide the users in the implementation stage
Telecommunication company			
• They concentrate on selected domains of ubiquitous computing	• Management of the system and decentralization of the technology development	• How intelligent should the system be? • Is it reasonable to build ubiquitous applications?	• Consumer electronics • Home environment • Entertainment

As can be seen from Table 4, the companies see ubiquitous software in very different ways, and the problems and opportunities they experience in the ubiquitous software area are also different. From this, it can be concluded that once the main issues of ubiquitous software, such as interoperability of the applications and standardization has been solved, the remaining challenges have to be separately considered from the viewpoint of the stakeholder roles.

4.7 Summary

This Chapter presented state-of-the-practise in Finnish companies based on replies received to interviews and questionnaire.

The companies of this research had various definitions for ubiquitous computing and background knowledge about the topic and this is one reason for differing views about existence of the ubiquitous technology. Some companies regard ubiquitous software as a new term for old things, they think that the ubiquitous technology already exists. On the other hand companies that regard ubiquitous software as a totally new idea, regard ubiquitous technology as non-existent. As a reason for lack of ubiquitous applications was seen:

1. lack of competent people,
2. lack of initiative (=someone should start the implementation),
3. lack of infrastructure, and basic services
4. weak quality level of current ubiquitous technologies.

The representatives of the companies regarded as main challenges of development of ubiquitous software:

- achieving standard interfaces and interoperability between applications and devices,
- enabling reusability of the components,
- standardization of software platforms and data received from different sources,
- achieving ubiquitous way of thinking,
- increased complexity of programs,
- successful combination of software and hardware, and
- achieving value chain for ubiquitous computing

The revenue logic of ubiquitous services is seen -according to the companies- unclear although the area has revenue potential and competent people. The services should be almost free to the consumer and money could be collected via advertisements as everything costs directly or indirectly.

5. Conclusions and recommendations

5.1 Future research in the ubiquitous software area

Table 5 summarizes the development trend in some key areas of ubiquitous software technologies. Technologies are categorized into four main classes, namely infrastructure, middleware, content and pervasive services and software development. Development includes methods, techniques and technologies used in each of the other three classes. Human-system interaction technologies are out of the scope of this research and that is why they are not considered in this summary. The summary is not exhaustive, but the selected topics are related to the topics described earlier in this report. The aim is to provide an understanding of how these topics will evolve in the future. The spans of evolution defined are medium and long-term. Medium-term means 5 years and long-term 10 years.

Table 5. Evolution of ubiquitous software technologies in the future.

Technology	Now	Medium-term	Long-term
Infrastructure			
Interoperability of heterogeneous networks	Manual	Automatic	Seamless
Standarized ontologies for profiles and services	Community support and management	Standarized ontologies and profile construction	Fusion of profiles across applications
Automatic resource discovery	Local networks		Wireless access networks
Knowledge based auto-configuration		Home and enterprise	Wireless access networks
Middleware			
Automatic and scalable service discovery	Prototypes	Enterprise, home	Personal networks
Adaptive resource management	Prototypes	Devices, systems	Personal networks
Interface standards for management of resources in heterogeneous environments	Prototypes	Initial specifications, prototypes	Specified
Authentication	Single sing-on, Smarcard, Javacard, VPN	Dynamic configuration, Tamper-proof, PKI in P2P	
Security methodologies (modeling and testing tools)	Limited	Specified	Automated
Design patterns	From generic to domain-specific patterns, i.e. pervasive middleware patterns		
Light-weight middleware	Prototypes	Standard (interoperability, heterogeneous environments), software solutions and applications	
Established standard compo-nents and contracts	Component types	Specified and in practice	

continues...

Content/ Pervasive services			
Dynamic filtering and transformation for adaptation	Prototype	Applications	Real-time solutions
Self-organizing software agents	Multi-agents	Self-organizing algorithms and strategies, negotiation techniques Self-services, Adaptive, intelligent self-configuration	
Development			
Model transformation		Prototypes	Specified
Dynamic architectures		Models, ontology-oriented design, light-weight solutions	
Architectural patterns	General patterns	Specific patterns	
Architecture quality and trade-off analysis methods	Initial versions	Systematic methods, tool support	Semi-automatic
Automated software testing	Protocol software	Adoption and adaptation to pervasive software development	

5.2 Opportunities for Finnish companies in the ubiquitous area

Finnish companies will have various opportunities in the ubiquitous area although some of these opportunities are dependent on the development of the required technology like sensors and infrastructure. In the ubiquitous software arena, the companies are seen to have business opportunities in the following areas:

- middleware components,
- ubiquitous computing environment and its components,
- sensors implemented locally to various conditions,
- small-sized applications and services,
- personalized services, and
- transfer into wireless communication,

Finnish companies also have a good chance to succeed in the ubiquitous arena internationally due to their strong knowledge of mobile technologies and embedded software technologies. With regard to business domains, there is potential in providing applications for home environment, industry automation, business and consumer electronics. Realizing the business opportunities of the ubiquitous arena requires tough removal of the interoperability problems and achieving general standardization into the area.

5.3 Recommendations

The aim of this section is to conclude recommendations that Finnish companies and research institutes and universities could take into account in future research and development. Table 6 presents these recommendations from a general, system, software and business point of view by describing the authors' view and view of the researched companies.

Table 6. Recommended research topics of ubiquitous software.

	Authors' view	**Companies' view**
Generic	– Security and privacy protection – Ontology-orientated design – Methods and tools for describing service and content semantics	– Data security, ownership and control – Information semantics – User interfaces and interaction of a user in different devices
Software	– Cost-efficient and dynamic networking and architecture solutions – Management of changing requirements – Dynamic architectures – Adaptive middleware services – A unified middleware standard – Service evaluation – Methods and tools for evaluating execution qualities – Methods how to add value to existing software solutions – Methods, tools and platforms for testing embedded product software	– Management of rapidly changing requirements – Defining software-intensive, tailorable, platform-independent product applications – Configuration of the functionality – Dynamic architectures – Middleware and use cases of ubiquitous applications – Component-based design – Defining standardized interfaces
System	– Interoperable distributed software platforms	– Defining services that are adaptable to changes in transmission networks, service level, terminal etc.
Business	– Business value chains	– Defining challenges and costs of the infrastructure implementation – Defining cost-effective and reasonable value chain – Defining consumer needs and potential markets; the needs could be collected with the aid of pre-study of potential applications and their advantages

Table 5 confirms that the most important research topics of ubiquitous software are interoperability and security of the services, defining services that are adaptable to changes in transmission networks, service level, terminal etc. and value chain for ubiquitous services. Interoperability and adaptability can be achieved by defining standardized interfaces, unified middleware and dynamic architectures.

References

Abrahamsson, P., Warsta, J., Siponen, M. T. & Ronkainen, J. 2003. New directions on agile methods: comparative analysis. In: proceedings of ICSE 2003. IEEE Computer Society: Los Alamitos. Pp. 244–254

Ailisto, H., Kotila, A. & Strömmer, E. 2003. Ubicom applications and technologies. VTT Technical Research Centre of Finland: Espoo. VTT Tiedotteita - Research Notes: 2201. ISBN 951-38-6154-6; 951-38-6155-4. 54 p.

Amin, M. 2000. Toward Self-Healing Infrastructure Systems. Computer, Aug. 2000, pp. 44–53.

Antoniac, P., Pulli, P., Kuroda, T., Bendas, D., Hickey, S. & Sasaki, H. 2002. Wireless User Perspectives in Europe: HandSmart Mediaphone Interface. Wireless Personal Communications, Vol. 22, September, pp. 161–174.

Bass, L., Clements, P. & Kazman, R. 1998. Software Architecture in Practice. Reading, Massachusetts, USA, Addison-Wesley.

Buschmann, F., Meunier, R., Rohnert, H. 1996. Pattern-oriented software architecture, as system of patterns. Chishester, John Wiley & Sons.

Bustamante, F. E., Widener, P. & Schwan, K. 2002. Scalable Directory Services Using Proactivity. Supercomputing 2002, Baltimore, MD.

Cabri, G., Leonardi, L. & Zambonelli, F. 2002. Engineering Mobile Agent Applications via Context-Dependent Coordination. IEEE Transactions on Software Engineering, Vol. 28, Issue 11, pp. 1039–1055.

Charles, J. 1999. Middleware moves to the forefront. Computer, Vol. 32, Issue 5, pp. 17–19.

Chiang, C. C. 2003. The use of adapters to support interoperability of components for reusability. IEEE Information and Software Technology, Vol. 45, pp. 149–156.

Clements, P. & Northrop, L. 2002. Software Product Lines: Practices and Patterns. Boston, MA, USA, Addison-Wesley.

Coulouris, G., Dollimore, J. & Kindberg, T. 2001. Distributed Systems Concepts and Design. Addison-Wesley. ISBN 0201-61918-0. 772 p.

Elder, M. C. 2001. Fault Tolerance in Critical Information Systems. Faculty of the School of Engineering and Applied Science, University of Virginia.

Ellison, R. J., Fisher, D. A., Linger, R. C., Lipson H. F., Longstaff, T. A. & Mead, N. R. 1999. An Approach to Survivable Systems, CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University.

Garlan, D., Allen, R. & Ockerbloom, D. 1995. Architectural Mismatch or Why It Is So Hard to Build Systems out of Existing Parts. The 17th International Conference on Software Engineering, Seattle, Washington, USA.

Gutberlet, L. 2000. Peer-to-Peer Computing- A Technology Fad or Fact? European Business School/Schloss Reichartshausen am Rhein.

Heylighen, F. & Gersheson, C. 2003. The meaning of Self-organization in Computing. IEEE Intelligent Systems. Section Trends & Controversies - Self-organization and Information Systems. May/Jun 2003.

Homayounfar, H. 2002. An advanced P2P architecture using autonomous agents. University of Guelph (Ontario, Canada) (Master's Thesis): 182 p.

Hongisto, M. 2003. Resource management under Linux for COTS and open source applications. Minttu technical report.

IBM. 2004. On-line at: http://www-106.ibm.com/developerworks/webservices/library/ws-wsxl/.

Kalaoja, J., Niemelä, E., Tikkala, A., Kallio, P., Ihme, T. & Torchiano, M. 2003. WISA Reference Architecture. Deliverable ID: D4 (Part B).

Kallio, P., Zorer, A. & Tiella, R. 2004. Accounting and billing of wireless Internet services in 3G networks. Accepted to International Journal of Mobile Communications.

Kindberg, T. & Fox, A. 2002. System Software for Ubiquitous computing. IEEE Pervasive computing January-March 2002.

Knight, J. C., Strunk, E. A. & Sullivan, K. J. 2003. Towards a Rigorous Definition of Information System Survivability. DARPA Information Survivability Conference and Exposition, DISCEX '03, Washigton DC.

Knight, J. C. & Sullivan, K. J. 2000. On the Definition of Survivability, University of Virginia, Department of Computer Science.

Kähkönen, T. & Abrahamsson, P. 2003. Digging into the fundamentals of extreme programming buildingthe theoretical base for agile methods. Proceedings of 29th Euromicro Conference, 2003, Antalaya, Turkey, 1–6 Sept. 2003. IEEE Computer Society. Los Alamitos (2003). Pp. 273–280.

Latvakoski, J. 1997. Integration Test Automation of embedded communication software. VTT Publication 318. 98 p. + app. 28 p.

Latvakoski, J., Pakkala, D. & Pääkkönen, P. 2004. A communication Architecture for Spontaneous Systems. IEEE Wireless Communication Magazine. 2004.

Matinlassi, M. & Niemelä, E. 2003. The Impact of Maintainability on Component-based Software Systems. Euromicro 2003, Antalya, Turkey.

Myers, G. J. 1979. The art of software testing. New York: John Wiley & Sons. 177 p.

Nakijima, T., Ishikawa, H., Tokunaga, E. & Stajano, F. 2002. Technology Challenges for Building Internet Scale Ubiquitous Computing. Proceedings of the 7th International Workshop on Object-oriented Real-Time Dependable Systems, WORDS 2002. Pp. 171–179.

Niemelä, E. & Vaskivuo, T. 2004. Agile Middleware of Pervasive Computing Environments. Middleware Support for Perwasive Computing Workshop, PerWare'2004, Orlando, USA, IEEE.

Niemelä, E., Matinalassi, M. & Lago, P. 2003. Architecture-centric approach to wireless service engineering. Annual Review of Communications, Volume 56. IEC. Chicago (2003). Pp. 875–889.

Noble, B. D., Narayannan, D., Tilton, J. E., Flinn, J. & Walker, K. R. 1997. Agile Application-Aware Adaptation for Mobility. The 16th ACM Symposium on Operating Systems Principles, Saint Malo, France, IEEE.

Nokia. 2002. Mobile Internet Technical Architecture. Volumes 1, 2 and 3. ISBN 951-826-668-9.

Pakkala, D. 2004. Lightweight Distributed Service Platform for Adaptive Mobile Services. MSc Thesis. University of Oulu. Department of Electrical and Information Engineering. 103 p.

Pal, P. P., Loyall, J. P., Schantz, R. E., Zinky, J. A. & Webber, F. 2000. Open implementation toolkit for building survivable applications. DARPA Information Survivability Conference and Exposition, DISCEX'00.

Parameswaran, M., Susarla, A. & Whinston, A. B. 2001. P2P Networking: An Information-Sharing Alternative. IEEE Computer. July 2001.

Perkins, C. E. (ed.). 2001. Ad Hoc Networking. Addison-Wesely. ISBN 0-201-30976-9.

Ponnekanti, S. R., Lee, B., Fox, A., Hanrahan, P. & Winograd, T. (eds). 2001. ICrafer: A Service Framework for Ubiquitous Computing Environments. Lecture Notes in Computer Science, Springer Verlag.

Purhonen, A. & Tuulari, E. 2003. Ambient Intelligence and the Development of Embedded System Software. Ambient Intelligence: Impact on Embedded System Design. T. Basten, M. Geilen and H. de Groot (eds.). Kluwer Academic Publishers**.** Pp. 51–67.

Raatikainen, K., Hohl, F., Latvakoski, J., Lindholm, T. & Tarkoma, S. 2003. Generic Service Elements for Adaptive Applications. WWRF WG2 White Paper. 12 p.

Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H. & Nahrstedt, K. 2002. A Middleware Infrastructure for Active Spaces. IEEE Pervasive Computing**,** Vol. 1, Issue 4, pp. 74–83.

Schulzrinne, H. & Rosenberg, J. 2000. Application layer mobility using SIP. ACM Sigmobile. Mobile Computing and Communications Review, Vol. 4, Issue 3. Jul 2000.

Sipilä, M. 2002. Communications Technologies. The VTT roadmaps. Espoo. VTT Research Notes 2146. 81 p.

Sousa, J. P. & Garlan, D. 2002. Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments. The 3rd working IEEE/IFIP Conference on Software Architecture, Montreal, Canada.

Tarvainen, P. 2004. Survey on Survivability of Information Systems. Submitted to ESORICS 2004, the 9th European Symposium on Research in Computer Security. 18 p.

Taulavuori, A., Niemelä, E. & Matinlassi, M. 2004. Evaluating the integrability of COTS components - the product family viewpoint. To be apper in: Building Quality into COTS Components - Testing and Debugging. S. Beydeda and V. Gruhn (Eds.), Springer-Verlag**.** 25 p.

Tierney, B., Aydt, R., Gunter, D., Smith, W., Taulor, V., Wolski, R. & Swany, M. 2002. A grid monitoring architecture, Global Grid Forum - Performance Working Group.

Toh, C.-K. 2002. Ad Hoc Mobile Wireless Networks. 302 p.

Weiser, M. 1991. The computer for the twenty-first century. Scientific American, Sep 1991, pp. 94–104.

Weiser, M. 1993. Some Computer Science issues in Ubiquitous Computing. Communications of the ACM, 36(7), Jul 1993.

Vinoski, S. 2002. Putting the "Web" into Web Services. IEEE Internet Computing, Vol. 6, Issue 4, pp. 90–92.

Vinoski, S. 2003. Service Discovery 101. IEEE Internet Computing, Vol. 7, Issue 1, pp. 69–71.

VTT Electronics. 2004. Embedded Software Research and Development Activities, to be appear Spring 2004.

WWRF. 2001. The Book of Visions 2001. Version 1.0. 281 p.

Xu, H. & Shatz, S. M. 2003. A Framework for Model-Based Design of Agent-Oriented Software. IEEE Transactions on Software Engineering, Vol. 29, Issue 1, pp. 15–30.

Zhou, J. & Niemelä, E. 2004a. Beyond Application-Oriented Software Engineering: Service-Oriented Software Engineering (SOSE). To be appear in: Service-Oriented Software System Engineering: Challenges and Practices. Z. Stojanovic and A. Dahanayake (Eds.). 22 p.

Zhou, J. & Niemelä, E. 2004b. Agile Software Development: A Survey and A Return on Experience. Submitted to ACM Computing Surveys, 14 p.

Published by

Series title, number and
report code of publication

VTT Research Notes 2238
VTT–TIED–2238

Author(s)
Kallio, Päivi, Niemelä, Eila & Latvakoski, Juhani

Title

Ubisoft- Pervasive Software

Abstract
Ubiquitous computing enhances computer use by making many computers available throughout the physical environment, while making them effectively invisible to the user. Ubiquitous computing can be seen as a prerequisite for pervasive computing that emphasizes mobile data access, and the mechanisms needed to support a community of nomadic users. Ambient intelligence focuses on a smart way to use communication technology for making life simpler, more enjoyable and interesting. Ubiquitous software is software required for ubiquitous computing environments and in this report it includes pervasive software. Ambience intelligence is out of the scope of this report. The aim of this report is to offer Finnish companies and preparators of the Tekes-programs (ELMO, NETS, FENIX) a total view of the maturity, development needs and business opportunities of software engineering in the ubiquitous computing area. This report illustrates the state-of-the-art and requirements of ubiquitous software based on recent surveys. This report also provides a view on the state-of-the-practice of ubiquitous software in Finnish companies based on interviews made in some Finnish companies and replies received to a questionnaire sent to bigger sample of companies. State-of-the-art visions set about global ubiquitous systems, products' short-time-to-market and quality requirements that are tightening up, set ubiquitous software a huge set of requirements that include interoperability, hetereogeneity, mobility, security, adaptability, ability of self-organization, augmented reality and scalable content. Enabling technologies of ubiquitous software are standards, reference architectures and generic software technologies. Ubiquitous software development requires applying suitable software architectures and development methods that are presented in this research. Based on the stat-of-the-practise this report presents that the main challenges of ubiquitous software are achieving adaptable middleware and interoperability between services and networks, developing the required enabling technologies, defining value chain for providing the services and guaranteeing secure transactions between different stakeholders. Ubiquitous computing is seen to combine hardware and software, so new kind of development methods and architectural models are required for ubiquitous service development. Companies will have business opportunities in ubiquitous business in middleware components, for example, concerning security, enabling technologies, ubiquitous components and sensors implemented locally to various conditions such as to the surface of paper, small-sized applications, and personalized services. In principal, we suggest as important research topics in ubiquitous software arena security, management of changing requirements, middleware standards and services, cost efficient architecture solutions and ubiquitous business value chains.

Keywords
ubiquitous software, pervasive computing, ubiquitous computing, ubiquitous business

Activity unit
VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland

ISBN
951–38–6452–9 (soft back ed.)
951–38–6453–7 (URL: http://www.vtt.fi/inf/pdf/)

Project number
E4SU00034

Date	Language	Pages	Price
April 2004	English, finnish abstr.	68 p.	B

Name of project
Ubisoft

Commissioned by
The National Technology Agency (Tekes)

Series title and ISSN
VTT Tiedotteita – Research Notes
1235–0605 (soft back edition)
1455–0865 (URL: http://www.vtt.fi/inf/pdf/)

Sold by
VTT Information Service
P.O.Box 2000, FIN–02044 VTT, Finland
Phone internat. +358 9 456 4404
Fax +358 9 456 4374

Julkaisija

Julkaisun sarja, numero ja raporttikoodi

VTT Tiedotteita 2238
VTT–TIED–2238

Tekijä(t)
Kallio, Päivi, Niemelä, Eila & Latvakoski, Juhani

Nimeke

Läsnä-älyn ohjelmistojen haasteet ja teknologiat

Tiivistelmä

Läsnä-älyn ohjelmistot lisäävät tietokoneiden käyttöä ja tarjoavat ohjelmistoilla tuotettuja palveluja käyttäjille hyödyntämällä käyttäjien normaalia fyysistä toimintaympäristöä, mutta kätkemällä tietokoneiden olemassaolon itse käyttäjiltä. Verkotettuja sulautettuja järjestelmiä tarvitaan, jotta käyttäjä saa haluamansa palvelut kaikissa mahdollisissa tilanteissa ja ympäristöissä. Tietokoneiden ja tietotekniikan leviäminen edellyttää myös tiedon saatavuutta liikkuvien päätelaitteiden kautta, mikä puolestaan edellyttää erityisiä liikkuvaa käyttäjää tukevia ratkaisuja. Älykkäät ympäristöt pyrkivät hyödyntämään tietoliikenneteknologiaa ihmisen elämän helpottamiseksi ja rikastuttamiseksi. Läsnä-älyn ohjelmisto tarkoittaa verkotettujen ja ympäristöön sulautettujen järjestelmien ohjelmistoa. Läsnä-älyn ohjelmistojen kehittäminen korostaa joko tietokonetekniikkaa tai ihmiskeskeisyyttä, joita molempia asioita on käsitelty tässä raportissa rinnakkain. Älykkäiden ympäristöjen kehittämiseen liittyvät teknologiat on jätetty raportin aihepiirin ulkopuolelle.

Tämän raportin tarkoitus on tarjota suomalaisille yrityksille ja Tekes-ohjelmien (ELMO, NETS, FENIX) valmistelijoille kokonaisnäkemys läsnä-älyn ohjelmistoteknologioiden kypsyydestä, kehitystarpeista, liiketoimintamahdollisuuksista ja ohjelmistokehityksen verkottumisesta sekä läsnä-älyn sovellusten mahdollisuuksista tulevaisuudessa. Raportti kuvaa läsnä-älyn ohjelmistojen teknologialle asettamat vaatimukset ja teknologioiden kypsyyden perustuen tuoreisiin tutkimustuloksiin. Raportti esittää myös läsnä-älyn sovelluksia kehittävien suomalaisten teollisuusyritysten käsityksen teknologian nykytilasta haastatteluihin ja kyselytutkimukseen perustuen.

Globaaleista langattomista kommunikointijärjestelmistä esitetyt visiot, palvelutuotteille asetettava lyhyt kehitysaika ja yhä kiristyvät laatuvaatimukset asettavat sekä järjestelmille että ohjelmistoille suuren joukon haasteita kuten liikkuvuus, yhteistoiminnallisuus ja mukautuvuus. Standardit, viitearkkitehtuurit ja yleiset ohjelmistoteknologiat ovat esimerkkejä teknologioista, jotka mahdollistavat läsnä-älyn ohjelmistojen kehittämisen. Koska kaikkialle leviävä tietojenkäsittely yhdistää ohjelmistot ja laitteistot, läsnä-älyn ohjelmistojen kehittämiseen tarvitaan uudenlaisia menetelmiä ja arkkitehtuurimalleja.

Tässä raportissa esitetään, että läsnä-älyn ohjelmistojen toteuttamisen suurimpia haasteita ovat mukautuvan välitason ohjelmistojen kehittäminen, ohjelmistopalvelujen ja kommunikointiverkkojen yhteistoiminnallisuuden aikaansaaminen, vaadittavan infrastruktuurin kehittäminen, toimivan arvoketjun määritteleminen palvelujen tuottamiseksi ja palvelujen turvallisen toimituksen takaaminen eri osapuolten välillä. Yrityksillä on liiketoimintamahdollisuuksia läsnä-älyn ohjelmistojen ja järjestelmien alueella mm. välitason komponenteissa, palvelut mahdollistavissa teknologioissa, sulautettujen järjestelmien komponenteissa, antureissa ja pienikokoisissa henkilökohtaistetuissa palveluissa.

Avainsanat
ubiquitous software, pervasive computing, ubiquitous computing, ubiquitous business

Toimintayksikkö
VTT Elektroniikka, Kaitoväylä 1, PL 1100, 90571 OULU

ISBN			Projektinumero
951–38–6452–9 (nid.) 951–38–6453–7 (URL: http://www.vtt.fi/inf/pdf/)			E4SU00034
Julkaisuaika Huhtikuu 2004	Kieli Englanti, suom. tiiv.	Sivuja 68 s.	Hinta B
Projektin nimi Ubisoft		Toimeksiantaja(t) Teknologian kehittämiskeskus (Tekes)	
Avainnimeke ja ISSN VTT Tiedotteita – Research Notes 1235–0605 (nid.) 1455–0865 (URL: http://www.vtt.fi/inf/pdf/)		Myynti: VTT Tietopalvelu PL 2000, 02044 VTT Puh. (09) 456 4404 Faksi (09) 456 4374	

VTT TIEDOTTEITA – RESEARCH NOTES

VTT ELEKTRONIIKKA – VTT ELEKTRONIK – VTT ELECTRONICS

1914 Korpipää, Tomi. Hajautusalustan suunnittelu reaaliaikasovelluksessa. 1998. 56 s. + liitt. 4 s.

1927 Lumpus, Jarmo. Kenttäväyläverkon automaattinen konfigurointi 1998. 68 s. + liitt. 3 s.

1933 Ihme, Tuomas, Kumara, Pekka, Suihkonen, Keijo, Holsti, Niklas & Paakko, Matti. Developing application frameworks for mission-critical software. Using space applications as an example. 1998. 92 p. + app. 20 p.

1965 Niemelä, Eila. Elektroniikkatuotannon joustavan ohjauksen tietotekninen infrastruktuuri. 1999. 42 s.

1985 Rauhala, Tapani. Javan luokkakirjasto testitapauseditorin toteutuksessa. 1999. 68 s.

2042 Kääriäinen, Jukka, Savolainen, Pekka, Taramaa, Jorma & Leppälä, Kari. Product Data Management (PDM). Design, exchange and integration viewpoints. 2000. 104 p.

2046 Savikko, Vesa-Pekka. EPOC-sovellusten rakentaminen. 2000. 56 s. + liitt. 36 s.

2065 Sihvonen, Markus. A user side framework for Composite Capability / Preference Profile negotiation. 2000. 54 p. + app. 4 p.

2088 Korva, Jari. Adaptiivisten verkkopalvelujen käyttöliittymät. 2001. 71 s. + liitt. 4 s.

2092 Kärki, Matti. Testing of object-oriented software. Utilisation of the UML in testing. 2001. 69 p. + app. 6 p.

2095 Seppänen, Veikko, Helander, Nina, Niemelä, Eila & Komi-Sirviö, Seija. Towards original software component manufacturing. 2001. 105 p.

2114 Sachinopoulou, Anna. Multidimensional Visualization. 2001. 37 p.

2129 Aihkisalo, Tommi. Remote maintenance and development of home automation applications. 2002. 85 p.

2130 Tikkanen, Aki. Jatkuva-aikaisten multimediasovellusten kehitysalusta. 2002. 55 s.

2157 Pääkkönen, Pekka. Kodin verkotettujen laitteiden palveluiden hyödyntäminen. 2002. 69 s.

2160 Hentinen, Markku, Hynnä, Pertti, Lahti, Tapio, Nevala, Kalervo, Vähänikkilä, Aki & Järviluoma, Markku. Värähtelyn ja melun vaimennuskeinot kulkuvälineissä ja liikkuvissa työkoneissa. Laskenta-periaatteita ja käyttöesimerkkejä. 2002. 118 s. + liitt. 164 s.

2162 Hongisto, Mika. Mobile data sharing and high availability. 2002. 102 p.

2201 Ailisto, Heikki, Kotila, Aija & Strömmer, Esko. Ubicom applications and technologies. 2003. 54 p.

2213 Lenkkeri, Jaakko, Marjamaa, Tero, Jaakola, Tuomo, Karppinen, Mikko & Kololuoma, Terho. Tulevaisuuden elektroniikan pakkaus- ja komponenttitekniikat. 2003. 78 s. + liitt. 4 s.

2238 Kallio, Päivi, Niemelä, Eila & Latvakoski, Juhani. UbiSoft - pervasive software. 2004. 68 p.

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT TIETOPALVELU PL 2000 02044 VTT Puh. (09) 456 4404 Faksi (09) 456 4374	VTT INFORMATIONSTJÄNST PB 2000 02044 VTT Tel. (09) 456 4404 Fax (09) 456 4374	VTT INFORMATION SERVICE P.O.Box 2000 FIN–02044 VTT, Finland Phone internat. + 358 9 456 4404 Fax + 358 9 456 4374

ISBN 951–38–6452–9 (soft back ed.) ISBN 951–38–6453–7 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1235–0605 (soft back ed.) ISSN 1455–0865 (URL: http://www.vtt.fi/inf/pdf/)