6

Architecting Embedded Software for Context-Aware Systems

Susanna Pantsar-Syväniemi
VTT Technical Research Centre of Finland
Finland

1. Introduction

During the last three decades the architecting of embedded software has changed by i) the ever-enhancing processing performance of processors and their parallel usage, ii) design methods and languages, and iii) tools. The role of software has also changed as it has become a more dominant part of the embedded system. The progress of hardware development regarding size, cost and energy consumption is currently speeding up the appearance of smart environments. This necessitates the information to be distributed to our daily environment along with smart, but separate, items like sensors. The cooperation of the smart items, by themselves and with human beings, demands new kinds of embedded software.

The architecting of embedded software is facing new challenges as it moves toward smart environments where physical and digital environments will be integrated and interoperable. The need for human beings to interact is decreasing dramatically because digital and physical environments are able to decide and plan behavior by themselves in areas where functionality currently requires intervention from human beings, such as showing a barcode to a reader in the grocery store. The smart environment, in our mind, is not exactly an Internet of Things (IoT) environment, but it can be. The difference is that the smart environment that we are thinking of does not assume that all tiny equipment is able to communicate via the Internet. Thus, the smart environment is an antecedent for the IoT environment.

At the start of the 1990s, hardware and software co-design in real time and embedded systems were seen as complicated matters because of integration of different modeling techniques in the co-design process (Kronlöf, 1993). In the smart environment, the co-design is radically changing, at least from the software perspective. This is due to the software needing to be more and more intelligent by, e.g., predicting future situations to offer relevant services for human beings. The software needs to be interoperable, as well as scattered around the environment, with devices that were previously isolated because of different communication mechanisms or standards.

Research into pervasive and ubiquitous computing has been ongoing for over a decade, providing many context-aware systems and a multitude of related surveys. One of those surveys is a literature review of 237 journal articles that were published between 2000 and

2007 (Hong et al., 2009). The review presents that context-aware systems i) are still developing in order to improve, and ii) are not fully implemented in real life. It also emphasizes that context-awareness is a key factor for new applications in the area of ubiquitous computing, i.e., pervasive computing. The context-aware system is based on pervasive or ubiquitous computing. To manage the complexity of pervasive computing, the context-aware system needs to be designed in new way—from the bottom up—while understanding the eligible ecosystem, and from small functionalities to bigger ones. The small functionalities are formed up to the small architectures, micro-architectures. Another key issue is to reuse the existing, e.g., communication technologies and devices, as much as possible, at least at the start of development, to minimize the amount of new things.

To get new perspective on the architecting of context-aware systems, Section two introduces the major factors that have influenced the architecting of embedded and real-time software for digital base stations, as needed in the ecosystem of the mobile network. This introduction also highlights the evolution of the digital base station in the revolution of the Internet. The major factors are standards and design and modeling approaches, and their usefulness is compared for architecting embedded software for context-aware systems. The context of pervasive computing calms down when compared to the context of digital signal processing software as a part of baseband computing which is a part of the digital base station. It seems that the current challenges have similarities in both pervasive and baseband computing. Section two is based on the experiences gathered during software development at Nokia Networks from 1993 to 2008 and subsequently in research at the VTT Technical Research Centre of Finland. This software development included many kinds of things, e.g., managing the feature development of subsystems, specifying the requirements for the system and subsystem levels, and architecting software subsystems. The research is related to enable context-awareness with the help of ontologies and unique micro-architecture.

Section three goes through the main research results related to designing context-aware applications for smart environments. The results relate to context modeling, storing, and processing. The latter includes a new solution, a context-aware micro-architecture (CAMA), for managing context when architecting embedded software for context-aware systems. Section four concludes this chapter.

2. Architecting real-time and embedded software in the 1990s and 2000s

2.1 The industrial evolution of the digital base station

Figure 1 shows the evolution of the Internet compared with a digital base station (the base station used from now on) for mobile networks. It also shows the change from proprietary interfaces toward open and Internet-based interfaces. In the 1990s, the base station was not built for communicating via the Internet. The base station was isolated in the sense that it was bound to a base station controller that controlled a group of base stations. That meant that a customer was forced to buy both the base stations and the base station controller from the same manufacturer.

In the 2000s, the industrial evolution brought the Internet to the base station and it opened the base station for module business by defining interfaces between modules. It also

dissolved the "engagement" between the base stations and their controllers as it moved from the second generation mobile network (2G) to third one (3G). Later, the baseband module of the base station was also reachable via the Internet. In the 2010s, the baseband module will go to the cloud to be able to meet the constantly changing capacity and coverage demands on the mobile network. The baseband modules will form a centralized baseband pool. These demands arise as smartphone, tablet and other smart device users switch applications and devices at different times and places (Nokia Siemens Networks, 2011).

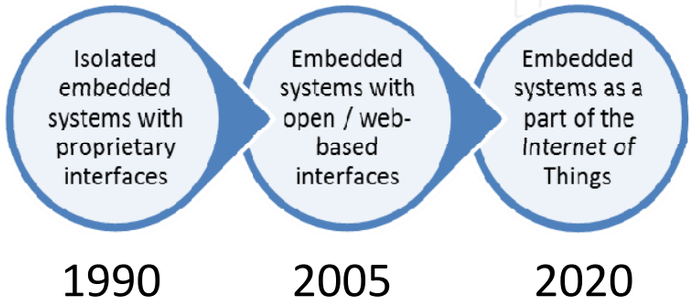


Fig. 1. The evolution of the base station.

The evolution of base-band computing in the base station changes from distributed to centralized as a result of dynamicity. The estimation of needed capacity per mobile user was easier when mobiles were used mainly for phone calls and text messaging. The more fancy features that mobiles offer and users demand, the harder it is to estimate the needed base-band capacity.

The evolution of the base station goes hand-in-hand with mobile phones and other network elements, and that is the strength of the system architecture. The mobile network ecosystem has benefited a lot from the system architecture of, for example, the Global System for Mobile Communications (GSM). The context-aware system is lacking system architecture and that is hindering its breakthrough.

2.2 The standardization of mobile communication

During the 1980s, European telecommunication organizations and companies reached a common understanding on the development of a Pan-European mobile communication standard, the Global System for Mobile Communications (GSM), by establishing a dedicated organization, the European Telecommunications Standards Institute (ETSI, www.etsi.org), for the further evolvement of the GSM air-interface standard. This organization has produced the GSM900 and 1800 standard specifications (Hillebrand, 1999). The development of the GSM standard included more and more challenging features of standard mobile technology as defined by ETSI, such as High Speed Circuit Switched Data (HSCSD), General Packet Radio Service (GPRS), Adaptive Multirate Codec (AMR), and Enhanced Data rates for GSM Evolution (EDGE) (Hillebrand, 1999).

The Universal Mobile Telecommunication System (UMTS) should be interpreted as a continuation of the regulatory regime and technological path set in motion through GSM, rather than a radical break from this regime. In effect, GSM standardization defined a path of progress through GPRS and EDGE toward UMTS as the major standard of 3G under the 3GPP standardization organization (Palmberg & Martikainen, 2003). The technological path from GSM to UMTS up to LTE is illustrated in Table 1. High-Speed Downlink Packet Access (HSDPA) and High-Speed Uplink Packet Access (HSUPA) are enhancements of the UMTS to offer a more interactive service for mobile (smartphone) users.

GSM -> HSCD, GPRS, AMR, EDGE	UMTS -> HSDPA, HSUPA	LTE
2G =>	3G =>	4G

Table 1. The technological path of the mobile communication system

It is remarkable that standards have such a major role in the telecommunication industry. They define many facts via specifications, like communication between different parties. The European Telecommunications Standards Institute (ETSI) is a body that serves many players such as network suppliers and network operators. Added to that, the network suppliers have created industry forums: OBSAI (Open Base Station Architecture Initiative) and CPRI (Common Public Radio Interface). The forums were set up to define and agree on open standards for base station internal architecture and key interfaces. This, the opening of the internals, enabled new business opportunities with base station modules. Thus, module vendors were able to develop and sell modules that fulfilled the open, but specified, interface and sell them to base station manufacturers. In the beginning the OBSAI was heavily driven by Nokia Networks and the CPRI respectively by Ericsson. Nokia Siemens Networks joined CPRI when it was merged by Nokia and Siemens.

The IoT ecosystem is lacking a standardization body, such as ETSI has been for the mobile networking ecosystem, to create the needed base for the business. However, there is the Internet of Things initiative (IoT-i), which is working and attempting to build a unified IoT community in Europe, www.iot-i.eu.

2.3 Design methods

The object-oriented approach became popular more than twenty years ago. It changed the way of thinking. Rumbaugh et al. defined object-oriented development as follows, i) it is a conceptual process independent of a programming language until the final stage, and ii) it is fundamentally a new way of thinking and not a programming technique (Rumbaugh et al., 1991). At the same time, the focus was changing from software implementation issues to software design. In those times, many methods for software design were introduced under the Object-Oriented Analysis (OOA) method (Shlaer & Mellor, 1992), the Object-Oriented Software Engineering (OOSE) method (Jacobson et al., 1992), and the Fusion method (Coleman et al., 1993). The Fusion method highlighted the role of entity-relationship graphs in the analysis phase and the behavior-centered view in the design phase.

The Object Modeling Technique (OMT) was introduced for object-oriented software development. It covers the analysis, design, and implementation stages but not integration and maintenance. The OMT views a system via a model that has two dimensions (Rumbaugh et al., 1991). The first dimension is viewing a system: the object, dynamic, or

functional model. The second dimension represents a stage of the development: analysis, design, or implementation. The object model represents the static, structural, "data" aspects of a system. The dynamic model represents the temporal, behavioral, "control" aspects of a system. The functional model illustrates the transformational, "function" aspects of a system. Each of these models evolves during a stage of development, i.e. analysis, design, and implementation.

The OCTOPUS method is based on the OMT and Fusion methods and it aims to provide a systematic approach for developing object-oriented software for embedded real-time systems. OCTOPUS provides solutions for many important problems such as concurrency, synchronization, communication, interrupt handling, ASICs (application-specific integrated circuit), hardware interfaces and end-to-end response time through the system (Awad et al., 1996). It isolates the hardware behind a software layer called the hardware wrapper. The idea for the isolation is to be able to postpone the analysis and design of the hardware wrapper (or parts of it) until the requirements set by the proper software are realized or known (Awad et al., 1996).

The OCTOPUS method has many advantages related to the system division of the subsystems, but without any previous knowledge of the system under development the architect was able to end up with the wrong division in a system between the controlling and the other functionalities. Thus, the method was dedicated to developing single and solid software systems separately. The OCTOPUS, like the OMT, was a laborious method because of the analysis and design phases. These phases were too similar for there to be any value in carrying them out separately. The OCTOPUS is a top-down method and, because of that, is not suitable to guide bottom-up design as is needed in context-aware systems.

Software architecture started to become defined in the late 1980s and in the early 1990s. Mary Shaw defined that i) architecture is design at the level of abstraction that focuses on the patterns of system organization which describe how functionality is partitioned and the parts are interconnected and ii) architecture serves as an important communication, reasoning, analysis, and growth tool for systems (Shaw, 1990). Rumbaugh et al. defined software architecture as the overall structure of a system, including its partitioning into subsystems and their allocation to tasks and processors (Rumbaugh et al., 1991). Figure 2 represents several methods, approaches, and tools with which we have experimented and which have their roots in object-oriented programming.

For describing software architecture, the 4+1 approach was introduced by Philippe Krüchten. The 4+1 approach has four views: logical, process, development and physical. The last view, the +1 view, is for checking that the four views work together. The checking is done using important use cases (Krüchten, 1995). The 4+1 approach was part of the foundation for the Rational Unified Process, RUP. Since the introduction of the 4+1 approach software architecture has had more emphasis in the development of software systems. The most referred definition for the software architecture is the following one:

> The structure or structures of the system, which comprises software elements, the externally visible properties of those elements, and the relationships among them, (Bass et al., 1998)

Views are important when documenting software architecture. Clements et al. give a definition for the view: "A view is a representation of a set of system elements and the

relationships associated with them". Different views illustrate different uses of the software system. As an example, a layered view is relevant for telling about the portability of the software system under development (Clements, 2003). The views are presented using, for example, UML model elements as they are more descriptive than pure text.

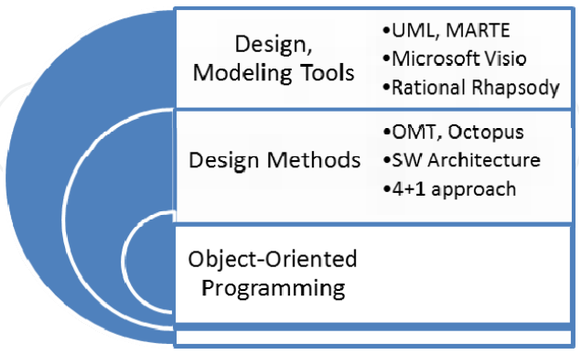


Fig. 2. From object-oriented to design methods and supporting tools.

Software architecture has always has a role in base station development. In the beginning it represented the main separation of the functionalities, e.g. operation and maintenance, digital signal processing, and the user interface. Later on, software architecture was formulated via architectural views and it has been the window to each of these main functionalities, called software subsystems. Hence, software architecture is an efficient media for sharing information about the software and sharing the development work, as well.

2.4 Modeling

In the model-driven development (MDD) vision, models are the primary artifacts of software development and developers rely on computer-based technologies to transform models into running systems (France & Rumpe, 2007). The Model-Driven Architecture (MDA), standardized by the Object Management Group (OMG, www.omg.org), is an approach to using models in software development. MDA is a known technique of MDD. It is meant for specifying a system independently of the platform that supports it, specifying platforms, choosing a particular platform for the system, and transforming the system specification into a particular platform. The three primary goals of MDA are portability, interoperability and reusability through the architectural separation of concerns (Miller & Mukerji, 2003).

MDA advocates modeling systems from three viewpoints: computational-independent, platform-independent, and platform-specific viewpoints. The computational-independent viewpoint focuses on the environment in which the system of interest will operate in and on the required features of the system. This results in a computation-independent model (CIM). The platform-independent viewpoint focuses on the aspects of system features that are not likely to change from one platform to another. A platform-independent model (PIM) is used to present this viewpoint. The platform-specific viewpoint provides a view of a system in which platform-specific details are integrated with the elements in a PIM. This view of a system is described by a platform-specific model (PSM), (France & Rumpe, 2007).

The MDA approach is good for separating hardware-related software development from the application (standard-based software) development. Before the separation, the maintenance of hardware-related software was done invisibly under the guise of application development. By separating both application- and hardware-related software development, the development and maintenance of previously invisible parts, i.e., hardware-related software, becomes visible and measurable, and costs are easier to explicitly separate for the pure application and the hardware-related software.

Two schools exist in MDA for modeling languages: the Extensible General-Purpose Modeling Language and the Domain Specific Modeling Language. The former means Unified Modeling Language (UML) with the possibility to define domain-specific extensions via profiles. The latter is for defining a domain-specific language by using meta-modeling mechanisms and tools. The UML has grown to be a de facto industry standard and it is also managed by the OMG. The UML has been created to visualize object-oriented software but also used to clarify the software architecture of a subsystem that is not object-oriented.

The UML is formed based on the three object-oriented methods: the OOSE, the OMT, and Gary Booch's Booch method. A UML profile describes how UML model elements are extended using stereotypes and tagged values that define additional properties for the elements (France & Rumpe, 2007). A Modeling and Analysis of Real-Time Embedded Systems (MARTE) profile is a domain-specific extension for UML to model and analyze real time and embedded systems. One of the main guiding principles for the MARTE profile (www.omgmarte.org) has been that it should support independent modeling of both software or hardware parts of real-time and embedded systems and the relationship between them. OMG's Systems Modeling Language (SysML, www.omgsysml.org) is a general-purpose graphical modeling language. The SysML includes a graphical construct to represent text-based requirements and relate them to other model elements.

Microsoft Visio is usually used for drawing UML–figures for, for example, software architecture specifications. The UML–figures present, for example, the context of the software subsystem and the deployment of that software subsystem. The MARTE and SysML profiles are supported by the Papyrus tool. Without good tool support the MARTE profile will provide only minimal value for embedded software systems.

Based on our earlier experience and the MARTE experiment, as introduced in (Pantsar-Syväniemi & Ovaska, 2010), we claim that MARTE is not as applicable to embedded systems as base station products. The reason is that base station products are dependent on long-term maintenance and they have a huge amount of software. With the MARTE, it is not possible to i) model a greater amount of software and ii) maintain the design over the years. We can conclude that the MARTE profile has been developed from a hardware design point of view because software reuse seems to have been neglected.

Many tools exist, but we picked up on Rational Rhapsody because we have seen it used for the design and code generation of real-time and embedded software. However, we found that the generated code took up too much of the available memory, due to which Rational Rhapsody was considered not able to meet its performance targets. The hard real-time and embedded software denotes digital signal processing (DSP) software. DSP is a central part of the physical layer baseband solutions of telecommunications (or mobile wireless) systems, such as mobile phones and base stations. In general, the functions of the physical

layer have been implemented in hardware, for example, ASIC (application-specific integrated circuits), and FPGA (field programmable gate arrays), or near to hardware (Paulin et al., 1997), (Goossens et al., 1997).

Due to the fact that Unified Modeling Language (UML) is the most widely accepted modeling language, several model-driven approaches have emerged (Kapitsaki et al., 2009), (Achillelos et al., 2010). Typically, these approaches introduce a meta-model enriched with context-related artifacts, in order to support context-aware service engineering. We have also used UML for designing the collaboration between software agents and context storage during our research related to the designing of smart spaces based on the ontological approach (Pantsar-Syväniemi et al., 2011a, 2012).

2.5 Reuse and software product lines

The use of C language is one of the enabling factors of making reusable DSP software (Purhonen, 2002). Another enabling factor is more advanced tools, making it possible to separate DSP software development from the underlying platform. Standards and underlying hardware are the main constraints for DSP software. It is essential to note that hardware and standards have different lifetimes. Hardware evolves according to 'Moore's Law' (Enders, 2003), according to which progress is much more rapid than the evolution of standards. From 3G base stations onward, DSP software has been reusable because of the possibility to use C language instead of processor-specific assembly language. The reusability only has to do with code reuse, which can be regarded as a stage toward overall reuse in software development, as shown in Figure 3.

Regarding the reuse of design outputs and knowledge, it was the normal method of operation at the beginning of 2G base station software developments and was not too tightly driven by development processes or business programs. We have presented the characteristics of base station DSP software development in our previous work (Pantsar-Syväniemi et al., 2006) that is based on experiences when working at Nokia Networks. That work introduces the establishment of reuse actives in the early 2000s. Those activities were development 'for reuse' and development 'with reuse'. 'For reuse' means development of reusable assets and 'with reuse' means using the assets in product development or maintenance (Karlsson, 1995).

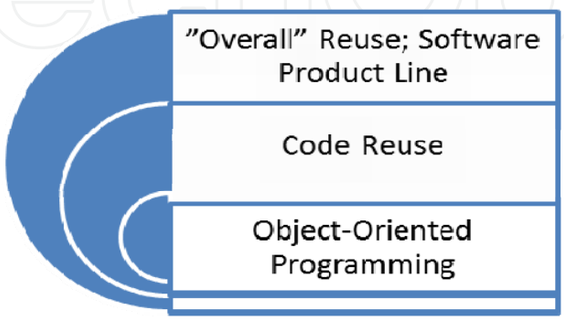


Fig. 3. Toward the overall reuse in the software development.

The main problem within this process-centric, 'for reuse' and 'with reuse', development was that it produced an architecture that was too abstract. The reason was that the domain was too wide, i.e., the domain was base station software in its entirety. In addition to that, the software reuse was "sacrificed" to fulfill the demand to get a certain base station product market-ready. This is paradoxical because software reuse was created to shorten products' time-to-market and to expand the product portfolio. The software reuse was due to business demands.

In addition to Karlsson's 'for and with reuse' book, we highlight two process-centric reuse books among many others. To design and use software architectures is written by Bosch (Bosch, 2000). This book has reality aspects when guiding toward the selection of a suitable organizational model for the software development work that was meant to be built around software architecture. In his paper, (Bosch, 1999), Bosch presents the main influencing factors for selecting the organization model: geographical distribution, maturity of project management, organizational culture, and the type of products. In that paper, he stated that a software product built in accordance with the software architecture is much more likely to fulfill its quality requirements in addition to its functional requirements.

Bosch emphasized the importance of software architecture. His software product line (SPL) approach is introduced according to these phases: development of the architecture and component set, deployment through product development and evolution of the assets (Bosch, 2000). He presented that not all development results are sharable within the SPL but there are also product-specific results, called artifacts.

The third interesting book introduces the software product line as compared to the development of a single software system at a time. This book shortly presents several ways for starting software development according to the software product line. It is written by Pohl et al. (Pohl et al., 2005) and describes a framework for product-line engineering. The book stresses the key differences of software product-line engineering in comparison with single-software system development:

- The need for two distinct development processes: domain engineering and application engineering. The aim of the domain-engineering process is to define and realize the commonality and the variability of the software product line. The aim of the application-engineering process is to derive specific applications by exploiting the variability of the software product line.
- The need to explicitly define and manage variability: During domain engineering, variability is introduced in all domain engineering artifacts (requirements, architecture, components, test cases, etc.). It is exploited during application engineering to derive applications tailored to the specific needs of different customers.

A transition from single-system development to software product-line engineering is not easy. It requires investments that have to be determined carefully to get the desired benefits (Pohl et al., 2005). The transition can be introduced via all of its aspects: process, development methods, technology, and organization. For a successful transition, we have to change all the relevant aspects, not just some of them (Pohl et al., 2005). With the base station products, we have seen that a single-system development has been powerful when products were more hardware- than software-oriented and with less functionality and complexity. The management aspect, besides the development, is taken into account in the

product line but how does it support long-life products needing maintenance over ten years? So far, there is no proposal for the maintenance of long-life products within the software product line. Maintenance is definitely an issue to consider when building up the software product line.

The strength of the software product line is that it clarifies responsibility issues in creating, modifying and maintaining the software needed for the company's products. In software product-line engineering, the emphasis is to find the commonalities and variabilities and that is the huge difference between the software product-line approach and the OCTOPUS method. We believe that the software product-line approach will benefit if enhanced with a model-driven approach because the latter strengthens the work with the commonalities and variabilities.

Based on our experience, we can identify that the software product-line (SPL) and model-driven approach (MDA) alike are used for base station products. Thus, a combination of SPL and MDA is good approach when architecting huge software systems in which hundreds of persons are involved for the architecting, developing and maintaining of the software. A good requirement tool is needed to keep track of the commonalities and variabilities. The more requirements, the more sophisticated tool should be with the possibility to tag on the requirements based on the reuse targets and not based on a single business program.

The SPL approach needs to be revised for context-aware systems. This is needed to guide the architecting via the understanding of an eligible ecosystem toward small functionalities or subsystems. Each of these subsystems is a micro-architecture with a unique role. Run-time security management is one micro-architecture (Evesti & Pantsar-Syväniemi, 2010) that reuses context monitoring from the context-awareness micro-architecture, CAMA (Pantsar-Syväniemi et al., 2011a). The revision needs a new mindset to form reusable micro-architectures for the whole context-aware ecosystem. It is good to note that micro-architectures can differ in the granularity of the reuse.

2.6 Summary of section 2

The object-oriented methods, like Fusion, OMT, and OCTOPUS, were dedicated for single-system development. The OCTOPUS was the first object-oriented method that we used for an embedded system with an interface to the hardware. Both the OCTOPUS and the OMT were burdening the development work with three phases: object-oriented analysis (OOA) object-oriented design (OOD), and implementation. The OOD was similar to the implementation. In those days there was a lack of modeling tools. The message sequence charts (MSC) were done with the help of text editor.

When it comes to base station development, the software has become larger and more complicated with the new features needed for the mobile network along with the UML, the modeling tools supporting UML, and the architectural views. Thus, software development is more and more challenging although the methods and tools have become more helpful. The methods and tools can also hinder when moving inside the software system from one subsystem to another if the subsystems are developed using different methods and tools.

Related to DSP software, the tight timing requirements have been reached with optimized C-code, and not by generating code from design models. Thus, the code generators are too

ineffective for hard real time and embedded software. One of the challenges in DSP software is the memory consumption because of the growing dynamicity in the amount of data that flows through mobile networks. This is due to the evolution of mobile network features like HSDPA and HSUPA that enable more features for mobile users. The increasing dynamicity demands simplification in the architecture of the software system. One of these simplifications is the movement from distributed baseband computing to centralized computing.

Simplification has a key role in context-aware computing. Therefore, we recall that by breaking the overall embedded software architecture into smaller pieces with specialized functionality, the dynamicity and complexity can be dealt with more easily. The smaller pieces will be dedicated micro-architectures, for example, run-time performance or security management. We can see that in smart environments the existing wireless networks are working more or less as they currently work. Thus, we are not assuming that they will converge together or form only one network. By taking care of and concentrating the data that those networks provide or transmit, we can enable the networks to work seamlessly together. Thus, the networks and the data they carry will form the basis for interoperability within smart environments. The data is the context for which it has been provided. Therefore, the data is in a key position in context-aware computing.

The MSC is the most important design output because it visualizes the collaboration between the context storage, context producers and context consumers. The OCTOPUS method is not applicable but SPL is when revised with micro-architectures, as presented earlier. The architecting context-aware systems need a new mindset to be able to i) handle dynamically changing context by filtering to recognize the meaningful context, ii) be designed bottom-up, while keeping in mind the whole system, and iii) reuse the legacy systems with adapters when and where it is relevant and feasible.

3. Architecting real-time and embedded software in the smart environment

Context has always been an issue but had not been used as a term as widely with regard to embedded and real-time systems as it has been used in pervasive and ubiquitous computing. Context was part of the architectural design while we created architectures for the subsystem of the base station software. It was related to the co-operation between the subsystem under creation and the other subsystems. It was visualized with UML figures showing the offered and used interfaces. The exact data was described in the separate interface specifications. This can be known as external context. Internal context existed and it was used inside the subsystems.

Context, both internal and external, has been distributed between subsystems but it has been used inside the base station. It is important to note that external context can be context that is dedicated either for the mobile phone user or for internal usage. The meaning of context that is going to, or coming from, the mobile phone user is meaningless for the base station but it needs memory to be processed. In pervasive computing, external context is always meaningful and dynamic. The difference is in the nature of context and the commonality is in the dynamicity of the context.

Recent research results into the pervasive computing state that:

- due to the inherent complexity of context-aware applications, development should be supported by adequate context-information modeling and reasoning techniques (Bettini et al., 2010)
- distributed context management, context-aware service modeling and engineering, context reasoning and quality of context, security and privacy, have not been well addressed in the Context-Aware Web Service Systems (Truong & Dustdar, 2009)
- development of context-aware applications is complex as there are many software engineering challenges stemming from the heterogeneity of context information sources, the imperfection of context information, and the necessity for reasoning on contextual situations that require application adaptations (Indulska & Nicklas, 2010)
- proper understanding of context and its relationship with adaptability is crucial in order to construct a new understanding for context-aware software development for pervasive computing environments (Soylu et al., 2009)
- ontology will play a crucial role in enabling the processing and sharing of information and knowledge of middleware (Hong et al., 2009)

3.1 Definitions

Many definitions for context as well for context-awareness are given in written research. The generic definition by Dey and Abowd for context and context-awareness are widely cited (Dey & Abowd, 1999):

'**Context** is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves. '

'**Context-awareness** is a property of a system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task. '

Context-awareness is also defined to mean that one is able to use context-information (Hong et al., 2009). Being context-aware will improve how software adapts to dynamic changes influenced by various factors during the operation of the software. Context-aware techniques have been widely applied in different types of applications, but still are limited to small-scale or single-organizational environments due to the lack of well-agreed interfaces, protocols, and models for exchanging context data (Truong & Dustdar, 2009).

In large embedded-software systems the user is not always the human being but can also be the other subsystem. Hence, the user has a wider meaning than in pervasive computing where the user, the human being, is in the center. We claim that pervasive computing will come closer to the user definition of embedded-software systems in the near future. Therefore, we propose that '*A context defines the limit of information usage of a smart space application*' (Toninelli et al., 2009). That is based on the assumption that any piece of data, at a given time, can be context for a given smart space application.

3.2 Designing the context

Concentrating on the context and changing the design from top-down to bottom-up while keeping the overall system in the mind is the solution to the challenges in the context-aware computing. Many approaches have been introduced for context modeling but we introduce one of the most cited classifications in (Strang & Linnhoff-Popien, 2004):

1. Key-Value Models

   The model of key-value pairs is the most simple data structure for modeling contextual information. The key-value pairs are easy to manage, but lack capabilities for sophisticated structuring for enabling efficient context retrieval algorithms.

2. Markup Scheme Models

   Common to all markup scheme modeling approaches is a hierarchical data structure consisting of markup tags with attributes and content. The content of the markup tags is usually recursively defined by other markup tags. Typical representatives of this kind of context modeling approach are profiles.

3. Graphical Model

   A very well-known general purpose modeling instrument is the UML which has a strong graphical component: UML diagrams. Due to its generic structure, UML is also appropriate to model the context.

4. Object-Oriented Models

   Common to object-oriented context modeling approaches is the intention to employ the main benefits of any object-oriented approach – namely encapsulation and reusability – to cover parts of the problems arising from the dynamics of the context in ubiquitous environments. The details of context processing are encapsulated on an object level and hence hidden to other components. Access to contextual information is provided through specified interfaces only.

5. Logic-Based Models

   A logic defines the conditions on which a concluding expression or fact may be derived (a process known as reasoning or inferencing) from a set of other expressions or facts. To describe these conditions in a set of rules a formal system is applied. In a logic-based context model, the context is consequently defined as facts, expressions and rules. Usually contextual information is added to, updated in and deleted from a logic based system in terms of facts or inferred from the rules in the system respectively. Common to all logic-based models is a high degree of formality.

6. Ontology-Based Models

   Ontologies are particularly suitable to project parts of the information describing and being used in our daily life onto a data structure utilizable by computers. Three ontology-based models are presented in this survey: i) Context Ontology Language (CoOL), (Strang et al., 2003); ii) the CONON context modeling approach (Wang et al., 2004); and iii) the CoBrA system (Chen et al., 2003a).

The survey of context modeling for pervasive cooperative learning covers the above-mentioned context modeling approaches and introduces a Machine Learning Modeling (MLM) approach that uses machine learning (ML) techniques. It concludes that to achieve the system design objectives, the use of ML approaches in combination with semantic context reasoning ontologies offers promising research directions to enable the effective implementation of context (Moore et al., 2007).

The role of ontologies has been emphasized in multitude of the surveys, e.g., (Baldauf et al., 2007), (Soylu et al., 2009), (Hong et al., 2009), (Truong & Dustdar, 2009). The survey related to context modeling and reasoning techniques (Bettini et al., 2010) highlights that ontological models of context provide clear advantages both in terms of heterogeneity and interoperability. Web Ontology Language, OWL, (OWL, 2004) is a de facto standard for describing context ontology. OWL is one of W3C recommendations (www.w3.org) for a Semantic Web. Graphical tools, such as Protégé and NeOnToolkit, exist for describing ontologies.

3.3 Context platform and storage

Eugster et al. present the middleware classification that they performed for 22 middleware platforms from the viewpoint of a developer of context-aware applications (Eugster et al., 2009). That is one of the many surveys done on the context-aware systems but it is interesting because of the developer viewpoint. They classified the platforms according to i) the type of context, ii) the given programming support, and iii) architectural dimensions such as decentralization, portability, and interoperability. The most relevant classification criteria of those are currently the high-level programming support and the three architectural dimensions.

High-level programming support means that the middleware platform adds a context storage and management. The three architectural dimensions are: (1) decentralization, (2) portability, and (3) interoperability. Decentralization measures a platform's dependence on specific components. Portability classifies platforms into two groups: portable platforms can run on many different operating systems, and operating system-dependent platforms, which can only run on few operating systems (usually one). Interoperability then measures the ease with which a platform can communicate with heterogeneous software components.

Ideal interoperable platforms can communicate with many different applications, regardless of the operating system on which they are built or of the programming language in which they are written. This kind of InterOperabilility Platform (IOP) is developed in the SOFIA-project (www.sofia-project.eu). The IOP's context storage is a Semantic Information Broker (SIB), which is a Resource Description Framework, RDF, (RDF, 2004) database. Software agents which are called Knowledge Processors (KP) can connect to the SIB and exchange information through an XML-based interaction protocol called Smart Space Access Protocol (SSAP). KPs use a Knowledge Processor Interface (KPI) to communicate with the SIB. KPs consume and produce RDF triples into the SIB according to the used ontology.

The IOP is proposed to be extended, where and when needed, with context-aware functionalities following 'the separation of concern' principle to keep application free of the context (Toninelli et al., 2009).

Kuusijärvi and Stenius illustrate how reusable KPs can be designed and implemented, i.e., how to apply 'for reuse' and 'with reuse' practices in the development of smart environments (Kuusijärvi & Stenius, 2011). Thus, they cover the need for programming level reusability.

3.4 Context-aware micro-architecture

When context information is described by OWL and ontologies, typically reasoning techniques will be based on a semantic approach, such as SPARQL Query Language for RDF (SPARQL), (Truong & Dustdar, 2009).

The context-awareness micro-architecture, CAMA, is the solution for managing adaptation based on context in smart environments. Context-awareness micro-architecture consists of three types of agents: context monitoring, context reasoning and context-based adaptation agents (Pantsar-Syväniemi et al., 2011a). These agents share information via the semantic database. Figure 4 illustrates the structural viewpoint of the logical context-awareness micro-architecture.

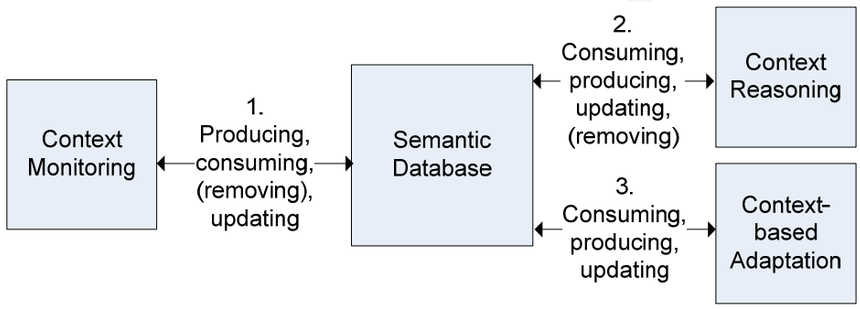


Fig. 4. The logical structure of the CAMA.

The context-monitoring agent is configured via configuration parameters which are defined by the architect of the intelligent application. The configuration parameters can be updated at run-time because the parameters follow the used context. The configuration parameters can be given by the ontology, i.e., a set of triples to match, or by a SPARQL query, if the monitored data is more complicated. The idea is that the context monitoring recognizes the current status of the context information and reports this to the semantic database. Later on, the reported information can be used in decision making.

The rule-based reasoning agent is based on a set of rules and a set of activation conditions for these rules. In practice, the rules are elaborated 'if-then-else' statements that drive activation of behaviors, i.e., activation patterns. The architect describes behavior by MSC diagrams with annotated behavior descriptions attached to the agents. Then, the behavior is transformed into SPARQL rules by the developer who exploits the MSC diagrams and the defined ontologies to create SPARQL queries. The developer also handles the dynamicity of the space by providing the means to change the rules at run-time. The context reasoning is a fully dynamic agent, whose actions are controlled by the dynamically changing rules (at run-time).

If the amount of agents producing and consuming inferred information is small, the rules can be checked by hand during the development phase of testing. If an unknown amount of agents are executing an unknown amount of rules, it may lead to a situation where one rule affects another rule in an unwanted way. A usual case is that two agents try to change the state of an intelligent object at the same time resulting in an unwanted situation. Therefore, there should be an automated way of checking all the rules and determining possible problems prior to executing them. Some of these problems can be solved by bringing

priorities into the rules, so that a single agent can determine what rules to execute at a given time. This, of course, implies that only one agent has rules affecting certain intelligent objects.

CAMA has been used:

- to activate required functionality according to the rules and existing situation(s) (Pantsar-Syväniemi et al., 2011a)
- to map context and domain-specific ontologies in a smart maintenance scenario for a context-aware supervision feature (Pantsar-Syväniemi et al., 2011b)
- in run-time security management for monitoring situations (Evesti & Pantsar-Syväniemi, 2010)

The Context Ontology for Smart Spaces, (CO4SS), is meant to be used together with the CAMA. It has been developed because the existing context ontologies were already few years old and not generic enough (Pantsar-Syväniemi et al, 2012). The objective of the CO4SS is to support the evolution management of the smart space: all smart spaces and their applications 'understand' the common language defined by it. Thus, the context ontology is used as a foundational ontology to which application-specific or run-time quality management concepts are mapped.

4. Conclusion

The role of software in large embedded systems, like in base stations, has changed remarkably in the last three decades; software has become more dominant compared to the role of hardware. The progression of processors and compilers has prepared the way for reuse and software product lines by means of C language, especially in the area of DSP software. Context-aware systems have been researched for many years and the maturity of the results has been growing. A similar evolution has happened with the object-oriented engineering that comes to DSP software. Although the methods were mature, it took many years to gain proper processors and compilers that support coding with C language. This shows that without hardware support there is no room to start to use the new methods.

The current progress of hardware development regarding size, cost and energy consumption is speeding up the appearance of context-aware systems. This necessitates that the information be distributed to our daily environment along with smart but separated things like sensors. The cooperation of the smart things by themselves and with human beings demands new kinds of embedded software. The new software is to be designed by the ontological approach and instead of the process being top-down, it should use the bottom-up way. The bottom-up way means that the smart space applications are formed from the small functionalities, micro-architecture, which can be configured at design time, on instantiation time and during run-time.

The new solution to designing the context management of context-aware systems from the bottom-up is context-aware micro-architecture, CAMA, which is meant to be used with CO4SS ontology. The CO4SS provides generic concepts of the smart spaces and is a common 'language'. The ontologies can be compared to the message-based interface specifications in the base stations. This solution can be the grounds for new initiatives or a body to start forming the 'borders', i.e., the system architecture, for the context-aware ecosystem.

5. Acknowledgment

The author thanks Eila Ovaska from the VTT Technical Research Centre and Olli Silvén from the University of Oulu for their valuable feedback.

6. References

Achillelos, A.; Yang, K. & Georgalas, N. (2009). Context modelling and a context-aware framework for pervasive service creation: A model-driven approach, *Pervasive and Mobile Computing*, Vol.6, No.2, (April, 2010), pp. 281-296, ISSN 1574-1192

Awad, M.; Kuusela, J. & Ziegler, J. (1996). *Object-Oriented Technology for Real-Time Systems. A Practical Approach Using OMT and Fusion*, Prentice-Hall Inc., ISBN 0-13-227943-6, Upper Saddle River, NJ, USA

Baldauf, M.; Dustdar, S. & Rosenberg, F. (2007). A survey on context-aware systems, *International Journal of Ad Hoc and Ubiquitous Computing*, Vol.2, No.4., (June, 2007), pp. 263-277, ISSN 1743-8225

Bass, L.; Clements, P. & Kazman, R. (1998). *Software Architecture in Practice*, first ed., Addison-Wesley, ISBN 0-201-19930-0, Boston, MA, USA

Bettini, C.; Brdiczka, O.; Henricksen, K.; Indulska, J.; Nicklas, D.; Ranganathan, A. & Riboni D. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, Vol.6, No.2, (April, 2010), pp.161—180, ISSN 1574-1192

Bosch, J. (1999). Product-line architectures in industry: A case study, *Proceedings of ICSE 1999 21st International Conference on Software Engineering*, pp. 544-554, ISBN 1-58113-074-0, Los Angeles, CA, USA, May 16-22, 1999

Bosch, J. (2000). *Design and Use of Software Architectures. Adopting and evolving a product-line approach*, Addison-Wesley, ISBN 0-201-67484-7, Boston, MA, USA

Chen, H.; Finin, T. & Joshi, A. (2003a). Using OWL in a Pervasive Computing Broker, *Proceedings of AAMAS 2003 Workshop on Ontologies in Open Agent Systems*, pp.9-16, ISBN 1-58113-683-8, ACM, July, 2003

Clements, P.C.; Bachmann, F.; Bass L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R. & Stafford, J. (2003). *Documenting Software Architectures, Views and Beyond*, Addison-Wesley, ISBN 0-201-70372-6, Boston, MA, USA

Coleman, D.; Arnold, P.; Bodoff, S.; Dollin, C.; Gilchrist, H.; Hayes, F. & Jeremaes, P. (1993). *Object-Oriented Development – The Fusion Method*, Prentice Hall, ISBN 0-13-338823-9, Englewood Cliffs, NJ, USA

CPRI. (2003). Common Public Radio Interface, 9.10.2011, Available from http://www.cpri.info/

Dey, A. K. & Abowd, G. D. (1999). *Towards a Better Understanding of Context and Context-Awareness*. Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, USA

Enders, A. & Rombach, D. (2003). *A Handbook of Software and Systems Engineering, Empirical Observations, Laws and Theories*, Pearson Education, ISBN 0-32-115420-7, Harlow, Essex, England, UK

Eugster, P. Th.; Garbinato, B. & Holzer, A. (2009) Middleware Support for Context-aware Applications. In: *Middleware for Network Eccentric and Mobile Applications* Garbinato, B.; Miranda, H. & Rodrigues, L. (eds.), pp. 305-322, Springer-Verlag, ISBN 978-3-642-10053-6, Berlin Heidelberg, Germany

Evesti, A. & Pantsar-Syväniemi, S. (2010). Towards micro architecture for security adaption, Proceedings of ECSA 2010 4th European Conference on Software Architecture Doctoral Symposium, Industrial Track and Workshops, pp. 181-188, Copenhagen, Denmark, August 23-26, 2010

France, R. & Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap. Proceedings of FOSE'07 International Conference on Future of Software Engineering, pp. 37-54, ISBN 0-7695-2829-5, IEEE Computer Society, Washington DC, USA, March, 2007

Goossens, G.; Van Praet, J.; Lanneer, D.; Geurts, W.; Kifli, A.; Liem, C. & Paulin, P. (1997) Embedded Software in Real-Time Signal Processing Systems: Design Technologies. *Proceedings of the IEEE*, Vol. 85, No.3, (March, 1997), pp.436–454, ISSN 0018-9219

Hillebrand, F. (1999). The Status and Development of the GSM Specifications, In: *GSM Evolutions Towards 3rd Generation Systems,* Zvonar, Z.; Jung, P. & Kammerlander, K., pp. 1-14, Kluwer Academic Publishers, ISBN 0-792-38351-6, Boston, USA

Hong, J.; Suh, E. & Kim, S. (2009). Context-aware systems: A literature review and classification. *Expert System with Applications*, Vol.36, No.4, (May 2009), pp. 8509-8522, ISSN 0957-4174

Indulska, J. & Nicklas, D. (2010). Introduction to the special issue on context modelling, reasoning and management, *Pervasive and Mobile Computing*, Vol.6, No.2, (April 2010), pp. 159-160, ISSN 1574-1192

Jacobson, I., et al. (1992). *Object-Oriented Software Engineering – A Use Case Driven Approach*, Addison-Wesley, ISBN 0-201-54435-0, Reading, MA, USA

Karlsson, E-A. (1995). *Software Reuse. A Holistic Approach*, Wiley, ISBN 0-471-95819-0, Chichester, UK

Kapitsaki, G. M.; Prezerakos, G. N.; Tselikas, N. D. & Venieris, I. S. (2009). Context-aware service engineering: A survey, *The Journal of Systems and Software*, Vol.82, No.8, (August, 2009), pp.1285-1297, ISSN 0164-1212

Kronlöf, K. (1993). *Method Integration: Concepts and Case Studies*, John Wiley & Sons, ISBN 0-471-93555-7, New York, USA

Krüchten, P. (1995). Architectural Blueprints—The "4+1" View Model of Software Architecture, *IEEE Software,* Vol.12, No.6, (November, 1995), pp.42-50, ISSN 0740-7459

Kuusijärvi, J. & Stenudd, S. (2011). Developing Reusable Knowledge Processors for Smart Environments, *Proceedings of SISS 2011 The Second International Workshop on "Semantic Interoperability for Smart Spaces" on 11th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2011),* pp. 286-291, Munich, Germany, July 20, 2011

Miller J. & Mukerji, J. (2003). MDA Guide Version 1.0.1. http://www.omg.org/docs/omg/03-06-01.pdf

Moore, P.; Hu, B.; Zhu, X.; Campbell, W. & Ratcliffe, M. (2007). A Survey of Context Modeling for Pervasive Cooperative Learning, *Proceedings of the ISITAE'07 1st IEEE International Symposium on Information Technologies and Applications in Education*, pp.K51-K56, ISBN 978-1-4244-1385-0, Nov 23-25, 2007

Nokia Siemens Networks. (2011). Liquid Radio - Let traffic waves flow most efficiently. White paper. 17.11.2011, Available from http://www.nokiasiemensnetworks.com/portfolio/liquidnet

OBSAI. (2002). Open Base Station Architecture Initiative, 10.10.2011, Available from http://www.obsai.org/

OWL. (2004). Web Ontology Language Overview, W3C Recommendation, 29.11.2011, Available from http://www.w3.org/TR/owl-features/

Palmberg, C. & Martikainen, O. (2003) *Overcoming a Technological Discontinuity - The case of the Finnish telecom industry and the GSM*, Discussion Papers No.855, The Research Institute of the Finnish Economy, ETLA, Helsinki, Finland, ISSN 0781-6847

Pantsar-Syväniemi, S.; Taramaa, J. & Niemelä, E. (2006). Organizational evolution of digital signal processing software development, *Journal of Software Maintenance and Evolution: Research and Practice*, Vol.18, No.4, (July/August, 2006), pp. 293-305, ISSN 1532-0618

Pantsar-Syväniemi, S. & Ovaska, E. (2010). Model based architecting with MARTE and SysML profiles. *Proceedings of SE 2010 IASTED International Conference on Software Engineering*, 677-013, Innsbruck, Austria, Feb 16-18, 2010

Pantsar-Syväniemi, S.; Kuusijärvi, J. & Ovaska, E. (2011a) Context-Awareness Micro-Architecture for Smart Spaces, *Proceedings of GPC 2011 6th International Conference on Grid and Pervasive Computing*, pp. 148–157, ISBN 978-3-642-20753-2, LNCS 6646, Oulu, Finland, May 11-13, 2011

Pantsar-Syväniemi, S.; Ovaska, E.; Ferrari, S.; Salmon Cinotti, T.; Zamagni, G.; Roffia, L.; Mattarozzi, S. & Nannini, V. (2011b) Case study: Context-aware supervision of a smart maintenance process, *Proceedings of SISS 2011 The Second International Workshop on "Semantic Interoperability for Smart Spaces", on 11th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2011),* pp.309-314, Munich, Germany, July 20, 2011

Pantsar-Syväniemi, S.; Kuusijärvi, J. & Ovaska, E. (2012) Supporting Situation-Awareness in Smart Spaces, *Proceedings of GPC 2011 6th International Conference on Grid and Pervasive Computing Workshops*, pp. 14–23, ISBN 978-3-642-27915-7, LNCS 7096, Oulu, Finland, May 11, 2011

Paulin, P.G.; Liem, C.; Cornero, M.; Nacabal, F. & Goossens, G. (1997). Embedded Software in Real-Time Signal Processing Systems: Application and Architecture Trends, *Proceedings of the IEEE,* Vol.85, No.3, (March, 2007), pp.419-435, ISSN 0018-9219

Pohl, K.; Böckle, G. & van der Linden, F. (2005). *Software Product Line Engineering*, Springer-Verlag, ISBN 3-540-24372-0, Berlin Heidelberg

Purhonen, A. (2002). *Quality Driven Multimode DSP Software Architecture Development*, VTT Electronics, ISBN 951-38-6005-1, Espoo, Finland

RDF. Resource Description Framework, 29.11.2011, Available from http://www.w3.org/RDF/

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F. & Lorensen, W. (1991) *Object-Oriented Modeling and Design*, Prentice-Hall Inc., ISBN 0-13-629841-9, Upper Saddle River, NJ, USA

Shaw, M. (1990). Toward High-Level Abstraction for Software Systems, *Data and Knowledge Engineering,* Vol. 5, No.2, (July 1990), pp. 119-128, ISSN 0169-023X

Shlaer, S. & Mellor, S.J. (1992) *Object Lifecycles: Modeling the World in States*, Prentice-Hall, ISBN 0-13-629940-7, Upper Saddle River, NJ, USA

Soylu, A.; De Causmaecker1, P. & Desmet, P. (2009). Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological

Engineering, *Journal of Software*, Vol.4, No.9, (November, 2009), pp.992-1013, ISSN 1796-217X

SPARQL. SPARQL Query Language for RDF, W3C Recommendation, 29.11.2011, Available from http://www.w3.org/TR/rdf-sparql-query/

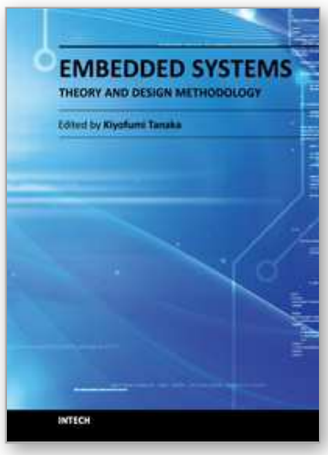
Strang, T.; Linnhoff-Popien, C. & Frank, K. (2003). CoOL: A Context Ontology Language to enable Contextual Interoperability, *Proceedings of DAIS2003 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, pp.236-247, LNCS 2893, Springer-Verlag, ISBN 978-3-540-20529-6, Paris, France, November 18-21, 2003

Strang, T. & Linnhoff-Popien, C. (2004). A context modelling survey, *Proceedings of UbiComp 2004 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pp.31-41, Nottingham, England, September, 2004

Toninelli, A.; Pantsar-Syväniemi, S.; Bellavista, P. & Ovaska, E. (2009) Supporting Context Awareness in Smart Environments: a Scalable Approach to Information Interoperability, *Proceedings of M-PAC'09 International Workshop on Middleware for Pervasive Mobile and Embedded Computing*, session: short papers, Article No: 5, ISBN 978-1-60558-849-0, Urbana Champaign, Illinois, USA, November 30, 2009

Truong, H. & Dustdar, S. (2009). A Survey on Context-aware Web Service Systems. *International Journal of Web Information Systems*, Vol.5, No.1, pp. 5-31, ISSN 1744-0084

Wang, X. H.; Zhang, D. Q.; Gu, T. & Pung, H. K. (2004). Ontology Based Context Modeling and Reasoning using OWL, *Proceedings of PerComW '04 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pp. 18–22, ISBN 0-7695-2106-1, Orlando, Florida, USA, March 14-17, 2004



Embedded Systems - Theory and Design Methodology
Edited by Dr. Kiyofumi Tanaka

ISBN 978-953-51-0167-3
Hard cover, 430 pages
Publisher InTech
Published online 02, March, 2012
Published in print edition March, 2012

Nowadays, embedded systems - the computer systems that are embedded in various kinds of devices and play an important role of specific control functions, have permitted various aspects of industry. Therefore, we can hardly discuss our life and society from now onwards without referring to embedded systems. For wide-ranging embedded systems to continue their growth, a number of high-quality fundamental and applied researches are indispensable. This book contains 19 excellent chapters and addresses a wide spectrum of research topics on embedded systems, including basic researches, theoretical studies, and practical work. Embedded systems can be made only after fusing miscellaneous technologies together. Various technologies condensed in this book will be helpful to researchers and engineers around the world.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen