Towards a unified middleware for ubiquitous and pervasive computing

F. J. Villanueva, F. Moya, M. J. Santofimia, F. Rincón, D. Villa, J. Barba, J. C. López
School of Computer Science, University of Castilla-La Mancha, Spain

***Abstract*—The broad variety of topics covered under the umbrella of *ubiquitous computing* led the research community to a fragmentation of the methods and tools used to achieve their goals. By contrast we believe that different goals do not necessarily mean different approaches. Since five years ago, we have been proponents of a unified approach based on the distributed object abstraction which allows synergies to be better exploited. Remote device management, wireless sensor and actuator networks, hardware components and now even reconfigurable hardware platforms and service platforms are considered as part of a large evolving system sharing a common middleware and a single design methodology. Considerable effort was put into the design of basic services to allow autonomous implementations on the smallest micro-controllers in the market.**

***Index Terms*—Design methodology, distributed computing, wireless sensor networks, integration architectures, object oriented methods.**

I. Introduction

Ubiquitous computing (UC) is all about heterogeneity, it deals with heterogeneity in heterogeneous ways. Some authors identify different areas of UC which lead to different approaches to building ubiquitous computing environments [1]. Sometimes there is an explicit distinction between ubiquitous computing and pervasive computing [2] to differentiate the main focus. Pervasive computing deals with providing adaptive or emerging services to fit user needs in a given context, while ubiquitous computing would be mainly focused on globally accessible services (anytime, anywhere).

In addition most research topics in UC must also deal with the fact that the target systems are inherently distributed. Therefore UC environments have much in common with distributed heterogeneous object platforms developed in the nineties (CORBA, EJB, DCOM, ...) and also with current grid computing platforms (Globus, gLite, ...). Our research tries to leverage the achievements of those platforms by defining a unified middleware able to interact with standard middlewares but specially suited to the needs of UC.

One major source of heterogeneity is the wide variety of devices connected by means of different networking technologies in almost every UC environment. The concept of residential gateway was coined in the field of residential services to mean a concentrator, a device with multiple network interfaces used to interconnect all the device networks available in a given scenario. Middlewares like OSGi [3] or Amigo [4] took advantage of this mediating device to provide a whole service management platform. Implicit to this approach is that most services will be running in the residential gateway. On one side this is positive from the point of view of manageability. The residential gateway may be controlled by the service provider or even by the telecommunications operator. But this device also constitutes a single point of failure which makes impractical the implementation of critical or very simple services (door opening, lighting, ...).

With more and more candidate technologies being integrated in ubiquitous computing environments it is increasingly harder to design a residential gateway at a reasonable cost, a single device with all required interfaces. From a business point of view, service providers are using residential gateways as a way to control the distribution of new services limiting users' freedom to choose alternative

Manuscript received July 6, 2008. This work was supported in part by the Spanish Government under Grants TIN2005-08719, and CENIT Hesperia and by Junta de Comunidades de Castilla-La Mancha under Grant PAI08-0234-8083.

All authors are with the Department of Information Technologies and Systems, UCLM, 13071 Ciudad Real, Spain (phone: +34 926-295-483; fax: +34 926-295-354; e-mail: francisco.moya@uclm.es).

service providers. While a mature market would see this as a competitive advantage, an emerging market such as the residential services sees a reduction of the perceived utility and consequently a slowdown in the rate of new deployments.

Currently DVB set-top-boxes, mobile phones, or public WiFi networks constitute alternative technologies for the interconnection of residential networks or device networks with an external service provider (e.g. through Internet). Therefore the overall architecture of an ubiquitous environment is evolving from a star topology centered around a residential gateway to a fully distributed mesh network where several devices provide some capabilities traditionally associated to a residential gateway.

As a conclusion, nowadays no single device will be able to support all the features required to provide services in a given scenario. Instead an increasing number of devices will need to cooperate in order to provide transparent gateway services among the different technologies.

This decentralized model also influences the design of the middleware:

- Management tasks (e.g. version control, start/stop commands, installation and removal of services, etc.) are distributed among a set of devices.
- The proper operation of the services in a UC environment does not depend on a single device. We may provide service replication procedures in order to increase the reliability.
- Security mechanisms will need to take into account the availability of several broadband and/or WAN interfaces (e.g. Internet).
- Service configuration is becoming increasingly complex.

We believe that object-oriented distributed middlewares include an attractive set of features to enable next generation ubiquitous computing environments. Six years ago we proposed an architecture based on the distributed object abstraction [5] to overcome the limitations of OSGi based large-scale residential services. This initial architecture was enhanced significantly in recent years to support highly dynamic environments, such as UC scenarios using heterogeneous RF communications [6][7][8][9]. Current work on this evolving architecture is mainly concerned with the remote deployment of distributed applications to low-end heterogeneous devices such as eight-bit microcontrollers (micro-components), and high-level support for service composition from semantic descriptions. Again we focus on minimalism in order to allow a certain degree of dynamic service discovery, aggregation and composition even for the smallest computing devices.

Distributed Object Based Services (DOBS) is the resulting framework after several iterations carried out by the ARCO Research Group at University of Castilla-La Mancha. DOBS aims at providing a full framework for ubiquitous computing built on top of standard distributed objects. Our main design goals were:

- Basic services should be easy to implement on any computing platform, no matter how small they are. Moreover, they must be able to be implemented in hardware.
- There must be support for remote service management.
- We look for maximum resource usage efficiency in terms of memory consumption, required bandwidth, etc.
- It should allow easy interoperability mechanisms in order to integrate third party services.
- It should provide a complete set of tools for ubiquitous computing application development and a set of templates for basic services. It should not rely on a specific implementation language or a particular operating systems.

The remainder of this paper is structured as follows. Section II describes the issues related to the lowest level of the architecture, built on wireless sensor and actuator networks. Section III

introduces the service concept as the core element of a middleware architecture aimed at supporting ubiquitous and pervasive computing. This section also outlines some of the key concepts involved in service composition, emphasizing the semantic model used to describe the application domain. Section IV is devoted to the specific considerations concerning the integration of WSN into our proposal in order to achieve a unified middleware. Finally, the last section exposes the designed workflow to generate hardware objects dedicated to those highly demanding tasks in terms of computational resources.

II. Wireless Sensor Networks

Besides the gradual transformation of infrastructures using residential gateways to fully distributed deployments, wireless sensor networks are also playing an important role in ubiquitous computing and influence the overall middleware design.

Wireless Sensor and Actuator Networks (WSANs) are becoming attractive for monitoring and small device control (door locks, windows, lighting, switches, etc.) due to their flexibility and reduced cost. Ambient intelligence could be the killer application WSANs are looking for.

Unit cost for WSAN devices is critical for the success of the technology, and the resources required for a middleware are directly translated into an increase of such unit cost. Therefore we feel that most of the middlewares specifically used in ubiquitous computing made the wrong decision when they chose either the Java platform or Web Services to deal with the heterogeneity. In our opinion these were made at the expense of larger resource requirements for target devices.

Similar considerations could be argued about textual message encodings such as XML as used in traditional Web Services or UPnP. Unlike conventional Ethernet or WiFi networks, devices in a WSAN are usually battery-powered and wireless communications drain most of the consumed power. Standard SOAP messages are less than efficient in a WSAN. On the other hand, most of the communication middlewares based on the distributed object abstraction (CORBA, ZeroC Ice, etc.) use compact binary protocols.

Obviously even these middlewares are too big for WSAN devices, but it is easier to trade-off flexibility for memory consumption. We developed a complete methodology to develop small distributed objects able to interact with standard middlewares [8]. While other approaches to embedding of these middlewares in resource constrained devices were focused on removing features from a standard middleware (e.g. [10]), we took the opposite approach building the minimum set of features to emulate the standard behavior.

For the simplest case (8-bit microcontrollers) we generate a custom state-machine to analyze messages produced by remote invocations and also to generate the response messages. The development of the software running on WSAN devices is quite similar to the development of any RMI based distributed application. We are developing a complete toolchain which generates simplified stubs and skeletons from interface descriptions, much like most standard middlewares based on the concept of distributed object.

III. Services and service composition

In DOBS, services are considered as the basic element of the system architecture. Furthermore, Ubiquitous and Pervasive Computing are service oriented and so they mainly deal with the way services are deployed, managed, or supported. In this context, services are not considered in an isolated manner, but in conjunction with the rest of services provided by the existing devices. In this regard, service composition provides the foundations for orchestrating existing services into composite ones.

Manual and semiautomatic composition, conversely to automatic composition, depends on user actions for accomplishing the composition task. Therefore, in an UC context where user interaction

tends to be minimized, automatic composition is the best solution to this end.

Automatic service composition can be achieved by means of web services and ontologies. These two approaches, however, lack the capabilities to deal with the high dynamism implicit in WSN,, where devices are continuously appearing and vanishing, and so are the services.

DOBS considers the combination of intelligent agents and ontologies as a mean for providing the middleware with reasoning capabilities. Among the existing solutions [11] to implement intelligent agents, the Belief-Desire-Intention (BDI) [12] model has proved to be a powerful framework to build rational agents. Furthermore, Jadex [13] provides an agent-oriented reasoning engine that once adapted to work on top of the middleware, provides the reasoning capabilities for the service composition mechanism in a dynamic context. In this regard, the ontology support is essential to understand and effectively react to changes and new requirements by means of automatic service composition.

A.Taxonomy representation

In order to define a common vocabulary and a basic set of services which both developers and manufacturers may find in any UC environment, we use a custom taxonomy with the following characteristics..

- Service names, their attributes and types are defined in a semi-formal way, and also the set of allowed values for the attributes.
- A common vocabulary is made available for any purpose. For example, it may be used to perform a search using the service discovery mechanisms with a set of attribute-value pairs.
- Relationships among services are described without limitations of simple tree-like structures.

We chose an ontology as the representation of our taxonomy because there are convenient tools available and it provides a flexible way of representing knowledge. Our application domain is reduced to ambient intelligence services though it may be extended easily.

The standardized set of services with known interfaces, names, attributes and allowed values for those attributes enables the development of advanced services.

The basic set of services described in the ontology was modeled after other common vocabularies from different sources, such as Bluetooth profiles, UPnP service templates, OMA Mobile Location Service, or OMG multimedia streaming service (AVStreams).

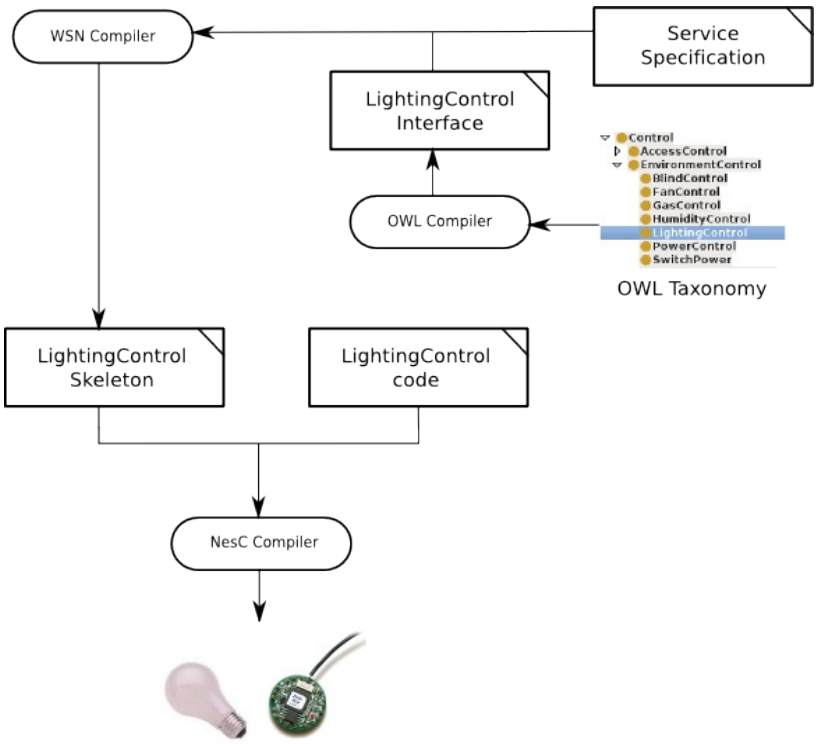

Figure 1: Design flow for DOBS objects on WSN devices.

The ontology was developed in OWL language using the Protegé tool. Services are modeled as classes and each class contains a set of attributes. An attribute has a type and, in some cases, a set of allowed values. The ontology by itself constitutes a valuable tool for a developer to get a quick understanding of the basic services. Besides, a compiler has been built to simplify the development even further (Figure 1). The ontology compiler generates interface descriptions for any service from the OWL description. Therefore the service developer is mainly concerned with the functional aspects of the service, with minimal considerations about remote interaction.

B.Place & Play philosophy

In ubiquitous computing environments, devices and services appear and disappear continuously. It may be because some of them are mobile devices or simply because devices and services may be turned on and off dynamically. Reducing configurations procedures is a key requirement for any practical environment. Assuming we have different devices using different remote services, it becomes clear that we will need gateways distributed around the environment in order to enable interoperability among services using heterogeneous technologies. But there are some additional requirements:

- A common addressing scheme for the services. Traditionally IP addresses fulfill this requirement but not every WSAN support a TCP/IP stack (not even for IP stacks with minimal resource requirements). Heterogeneous gateways are more conveniently handled with a routing component at the application layer.
- A service discovery protocol providing lookup and announce services. This service discovery protocol (SDP) must be available even for the smallest services and there must be interoperability procedures with other SDP since it is not feasible to impose a unique SDP. In DOBS we provide a minimum set of primitives which may be implemented even in a WSAN device. DOBS service discovery service may implement either a push or a pull mode and it is already being integrated with different SDP (UPnP SSDP, and Bluetooth).
- A bootstrap service used to initialize a device and integrate it in the environment. Most of the services to be integrated in a ubiquitous computing environment will need other basic services.

The bootstrap service may be used to get initial references for such basic services. For example, the DOBS SDP uses an event-oriented model to announce services or to search services. Therefore one of the first tasks needed to integrate a service in a DOBS enabled ubiquitous computing is getting a reference to the event channel service.

C.Standards

The lack of standards in ubiquitous environments is an obstacle that prevents these applications from leaving the research labs and reaching the mass market. In DOBS we follow existent standards as much as possible applying systematic rules when these standards need to be adapted somehow. Systematic translation of standard interfaces allows easier integration with third-party implementations of the standard. For example, we use OMG standards but the interface descriptions are translated into the DOBS interface description language. Currently we use the following service standards:

- For Audio and Video streams we use OMG AVStreams specification.
- For position related services we adopted the Mobile Location Protocol from OMA and a partial OpenLS specification from the OpenGIS Consortium.
- For basic device services the taxonomy was designed after UPnP templates and Bluetooth profiles.

IV.Hardware nodes

Within the framework of WSNs, there is a wide variety of nodes that conforms the system. The use of DOBS within WSNs is primary intended to encapsulate the nodes with less computational and memory requirements into an object-like appearance. Due to technological restrictions, those applications that needs of high computational resources to achieve hard time deadlines must move on to a hardware implementation.

For example, imagine a special node that must perform some kind of image processing algorithm to detect movement in a secure environment. This node must notify the object controller when a possible security breach is taking place.

Along with the automation of software embedded objects for small microcontrollers, we have developed a parallel design flow to easily create, integrate and interoperate with hardware objects. A hardware object (from now on HwO) is a custom integrated circuit that will perform a complex task in a WSNs.

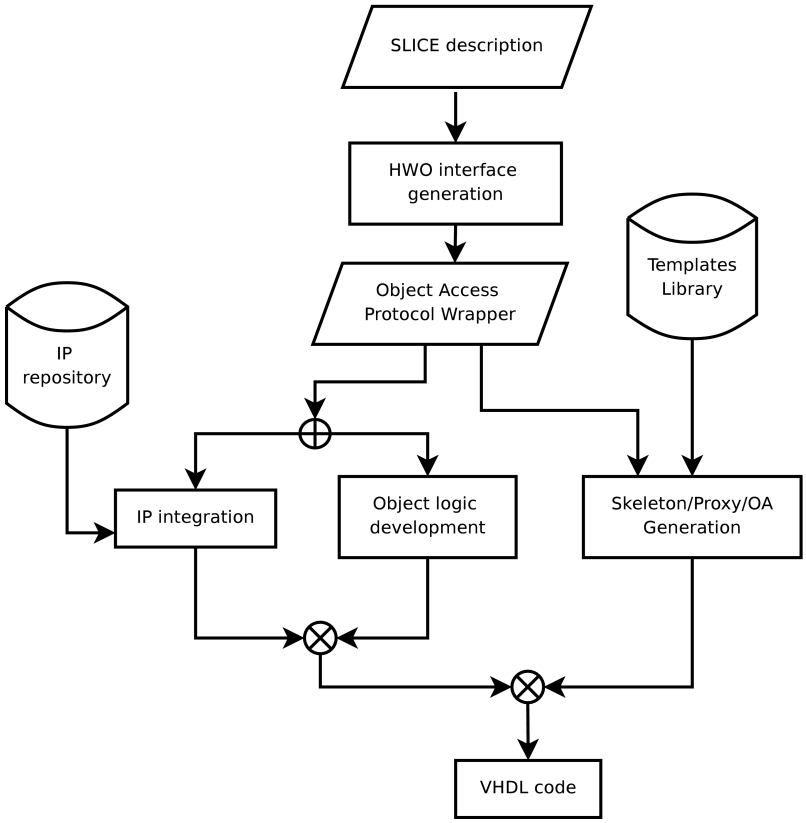

Figure 2: Design flow for DOBS objects on WSN devices.

Figure 2 shows the workflow of the proposed HwO generation process. The input to this process is an object interface description, written in an interface description language, just like the other objects will see the hardware component in the system. As we will see below, a hardware object exhibits a distinctive interface that determines how its functionality is accessed (object access protocol). This information is used to generate: (1) the on-chip communication infrastructure and (2) the final hardware object implementation. This HwO implementation can be the result of a integration process, reusing existing designs implementing the required functionality. A new, hand coded chip implementation is needed when no reuse opportunities are present for the current application. Notice that this process is completely automatic. finally, the resulting VHDL (a hardware description language) code is then synthesized.

A WSN is modeled as a set of objects that communicates through message passing. Concepts as attributes, method, parameters are mature in the software community since they have been widely applied in the development of any kind of projects for decades; this is not the case in hardware. It is then mandatory to establish a hardware object model to reduce the existing gap between the two worlds (hardware and software).

The benefits of viewing hardware components as objects is twofold since (1) leaves invariable the system model making transparent the integration of software and hardware components and (2) enables the automatic generation of the interconnection infrastructure because of the introduction of semantics in the communication into and out of the chip.

To implement logical objects as physical hardware components, we define: (1) and standardized interface in order to automatize the generation of wrappers and (2) a method invocation mechanism that must be understood by it. A hardware object exhibits a distinctive physical interface:

- Common system signals (clock, reset, etc.).
- One input signal per method to activate it.
- One output signal per method to indicate the end of an operation. This is used for synchronization

purposes.

- Several input/output data ports.

Our hardware object model uses the same data type system as the software implementation in order to define the inter-component communication semantics. At the time of writing this paper, all basic types (bool, short, int, float, etc.) plus structures, sequences (vectors) of a fixed size and any combination of them are supported by HwOs.

Figure 2 illustrates the module interfaces for a object implementing a data buffer of integers. We developed the HwO Access Protocol (HAP) tool that generates (1) scheduling information about the hardware method invocation protocol and (2) VHDL code of the finite state machine (FSMs) that understands the remote invocation protocol.

A.Hardware Object Platform

A HwO only implements the functionality it was designed for. We intentionally decoupled communication from behavior to make HwO modules reusable in future designs. By isolating HwOs from communication implementation details we make them immune to unforeseen changes in the communication infrastructure. To make HwOs accessible from outside the chip, we define a hardware object platform.

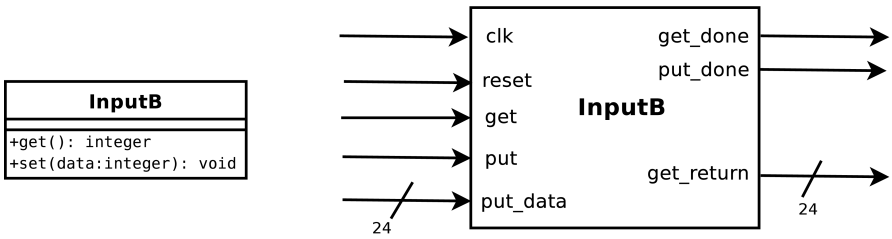

Figure 3: Design flow for DOBS objects on WSN devices.

Several HwOs can be grouped together into a single HwO platform to share resources and reduce implementation cost. Each hardware object in the system has an unique identifier that differs from the OID. Actually, this identifier corresponds with the base address of the module implementing the skeleton. The HwOA is responsible for translating external OID to internals valid bus addresses. Both proxies and skeletons agree in how method invocations translate in a sequence of low level actions over the bus to activate the execution of the operation. The data encoding/decoding rules are the same as in software. Therefore a method call is decomposed into write and read primitives. Read and write are basic services offered by most of the buses so that we do not limit the platform implementation to a concrete technology.

B.Interconnection Infrastructure Generation

The generation of skeletons (the hardware object wrappers) and the HwOA which implies the generation of proxies is a critical step in the hardware design flow. The final implementation must be highly optimized to fit the low-cost requirements while keeping the process automatic to save design time.

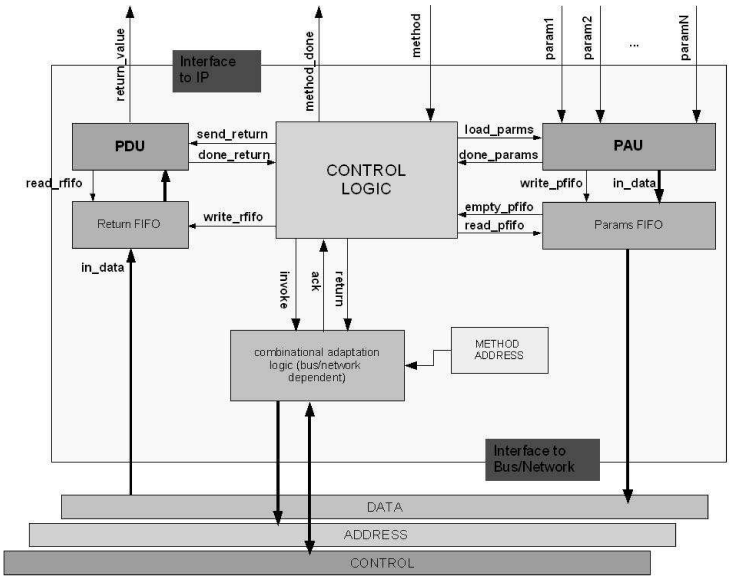

Figure 4: Internal architecture of a hardware proxy.

We defined two interface templates which will be specialized according to the number and kind of methods to be interpreted. Figure 4 shows the general architecture of a proxy.

The most important modules are the Port Acquisition Units (PAU) and the Port Delivery Unit (PDU) that are responsible for the adaptation of the data received (HwO or Bus) according to the object access protocol imposed by the HwO interface.

The Combinational Adaptation Logic (CAL) translates the control signals into specific bus commands and viceversa. The correspondence between FSM control signals and bus control signals is defined separately in system configuration files. The Control Logic implements the FSM doing the local (object access) and remote protocol (bus communication) adaptation. Eight parameterized VHDL design modules has been developed (one for each case and actor in the communication) .

A skeleton is only produced for those hardware objects exporting at least one method. Only methods remotely invoked by the client, using a proxy to the target is generated. This reduces the logic used in the proxy since there is no implementation for unused methods.

An additional optimization introduced in this process is the reutilization of logic when two method definitions not necessary belonging to the same object/class match (identical physical hardware object interface).

V. Conclusions and future work

Ubiquitous computing must deal with heterogeneity and with constrained resources on some computing nodes. The research community handled these issues with domain specific middlewares taking advantage of the features of the target platform. But we showed that using the abstraction of the distributed object implemented in many current general purpose middlewares we may better leverage the tools and services of standard middlewares. Constrained devices such as WSN nodes or hardware modules may use an adaptation of the standard design flow with minimal impact on the application programmer.

While the application-level interface is completely uniform across the whole system this is not made at the expense of preventing customized behavior for optimum performance. Users may provide additional transport protocols for new communication technologies or they may even implement critical services on custom hardware. Location transparency is preserved as much as possible.

Besides, the distributed object model makes some advanced services a feasible goal. Two specially

important examples are object persistency and object migration. Object persistency is the ability to store the internal state of the object in a persistent store and restore it when needed. This is extremely useful to decouple the life-time of a service from the life-time of the program implementing it. Services, even a small sensor service may be frozen in a storage device (e.g. a battery powered memory, a flash memory, etc.) when not needed and re-instantiated afterwards. Serialization of the internal state is completely analogous to serialization of parameters in remote method invocations. Therefore it is essentially free.

We may even go further if we restore an object from a persistent storage service to a different device. With additional support from a lightweight transaction service to guarantee atomicity this is essentially a dynamic migration service. It is even possible to migrate a service from a software implementation to a hardware implementation at run-time while it is being used by other components of the system.

References

[1] C. Endres, A. Butz, A. MacWilliams, *A Survey of Software Infrastructure and Frameworks for Ubiquitous Computing*, Mobile Information Systems, vol. 1, num. 1, pp. 41-80, IOS Press, 2005.

[2] J. Gaber, *Spontaneous Emergence Model for Pervasive Environments*, Proc. of Globecom Workshops, pp. 1-4, IEEE, Washington, DC, USA, Nov. 2007.

[3] The OSGi Alliance, *OSGi Service Platform: Core Specification, version 4.0.1*, www.osgi.org, July 2006.

[4] N. Georgantas, S. B. Mokhtar, Y.D. Bromberg, V. Issarny, J. Kalaoja, J. Kantarovitch, A. Gérodolle, R. Mevissen, *The Amigo Service Architecture for the Open Networked Home Environment*, Proc. of 5th Working IEEE/IFIP Conference on Software Architecture, pp. 295-296, IEEE, Pittsburgh, Nov. 2005.

[5] F. Moya J.C. Lopez, *SENDA: An Alternative to OSGi for Large Scale Domotics*, Proc. of IEEE Networks Conference, pp. 165-176, World Scientific, Atlanta, Jan 2002.

[6] D. Villa, F. J. Villanueva, F. Moya, F. Rincon, J. Barba, J. C. Lopez, *Embedding a Middleware for Networked Hardware and Software Objects*, Advances in Grid and Pervasive Computing, First International Conference, GPC 2006, Proceedings, pp. 567-576, Springer, Taichung, Taiwan, May 2006.

[7] D. Villa, F. J. Villanueva, F. Moya, J. Barba, F. Rincon, J. C. Lopez, *Minimalist Object Oriented Service Discovery Protocol for Wireless Sensor Networks*, Advances in Grid and Pervasive Computing, Second International Conference, GPC 2007, Proceedings, pp. 472-483, Springer, Paris, May 2007.

[8] F. Moya, D. Villa, F. J. Villanueva, J. Barba, F. Rincon, J. C. Lopez, J. Dondo, *Embedding Standard Distributed Object-Oriented Middlewares in Wireless Sensor Networks*, Wireless Communications and Mobile Computing Journal, vol. 12, num. 3, pp. 315-327, Wiley, Mar. 2007.

[9] F. J. Villanueva, D. Villa, F. Moya, J. Barba, F. Rincon, J. C. Lopez, *Lightweight Middleware for Seamless HW-SW Interoperability, with Applications to Wireless Sensor Networks*, Proc. of Design, Automation and Test in Europe, DATE 2007, pp. 1042-1047, IEEE, Nice Acropolis, France, April 2007.

[10] V. Subramonian, G. Xiang, *Middleware Specification for Memory-Constrained Networked Embedded Systems*, Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 306-313, IEEE, , May 2004.

[11] M. Wooldridge, *Reasoning about Rational Agents*, Intelligent robotics and autonomous agents series, vol. 16, num. , 227, The MIT Press, 2000.

[12] A. S. Rao, M. P. Georgeff, *Modeling rational agents within a BDI-architecture*, Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91), vol. 10, num. 3, 473-484, Morgan Kaufmann publishers Inc., .

[13] A. Pokahr, L. Braubach, W. Lamersdorf, *Jadex: A BDI Reasoning Engine*, R. Bordini, M. Dastani, J. Dix and A. El Fallah Seghrouchni (Hrsg.): Multi-Agent Programming, vol. 3, num. 3, 149-174, Springer Science+Business Media Inc., 2005.