Modeling Situation-Aware Ambient Assisted Living Systems for Eldercare

Werner Kurschl
Upper Austria University of Applied Sciences
School of Informatics,
Communications and Media
Department of Software Engineering
Softwarepark 11, 4232 Hagenberg, Austria

Stefan Mitsch and Johannes Schoenboeck
Upper Austria University of Applied Sciences
Research Center Hagenberg
Softwarepark 11, 4232 Hagenberg, Austria

Abstract

The development of ambient assisted living (AAL) systems, which are tailored to health or elder care, requires specific methods and tools. AAL systems make often use of wireless sensor networks, machine learning algorithms and sensory devices. Since wireless sensor networks and their sensors are inhomogeneous, it became apparent that such systems need to cope with different hardware platforms, different programming languages, unreliable wireless communication, energy constraints, data analysis algorithms, recognition of situations, and deployment options. Developers to date tend to use a bottom-up approach: hardware components dictate the development of AAL systems and thereby restrict the range of use cases that can be realized; domain experts by contrast would prefer a top-down approach and model the system's functionality independently from the hardware platform. Currently available software development environments and tools do not adequately support domain experts and developers to accomplish these tasks efficiently.

This paper presents methods that support domain experts in their top-down approach, as well as technically experienced developers in their bottom-up approach. The implemented tools enable a model-driven software development process (from platform-independent modeling to generating AAL application code) and thus facilitate programming AAL systems.

Index Terms
AAL, MDSD, Wireless Sensor Networks.

1. Introduction

Ambient assisted living (AAL) systems aim at improving the quality of life of elder or otherwise needy persons. In particular AAL systems should extend the time people can live in their homes, increase their autonomy and mobility, help them maintain better health, enhance their security, and support care organizations and families (see [18]). AAL systems are seamlessly embedded into the environment; they acquire knowledge of their surroundings with sensors, automatically and autonomously act on behalf of users, or they interact with users in a non-obtrusive and intuitive way. Current research projects focus on implementing prototypes for single aspects or use cases of an AAL system (e.g., a fall detection application). By comparing different prototypes we discovered that many of them use similar processing steps and implementation details:

1. Sensors acquire environmental and vital parameters (raw sensor data, preprocessed features or low-level context data) from their surroundings.
2. Inference engines interpret these data and derive high-level semantics, also called situations, which recursively can be input again to inference engines.
3. Semantic reasoning algorithms—e.g., rules engines—select an action from several possible ones if a particular situation is detected (they make decisions).
4. Actuators initiate the selected action (e.g., raise an alarm).

We therefore defined in [9] a reference architecture for AAL systems, which is closely related to the works of [21] and [1]. Our reference architecture was originally structured into three deployment tiers. These deployment boundaries proved to be too restrictive, because today's sensor platforms (e.g., Crossbow MICAz) are not only able to acquire data, but also feature enough processing power and memory to execute situation inference algorithms (so-called in network processing, see [16]); hence, we cannot assume fixed deployment boundaries between the logical layers in the reference architecture. The refined architecture is depicted in figure 1.

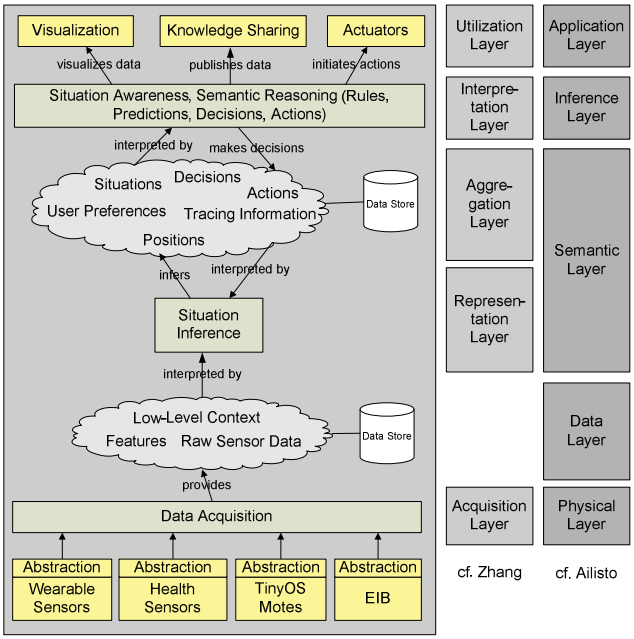

Figure 1. AAL system reference architecture.

The following problems are inherent in the design and implementation of AAL systems. Use cases are typically implemented bottom-up, starting with the selection of sensor hardware (because sensors are specialized to measure only certain phenomena). The sensor hardware then dictates the remaining development process (e.g., the programming language and the communication protocols being used). This results in use cases being tightly coupled to the selected hardware platform; thus, poor reusability, poor maintainability, poor scalability, and platform dependency are possible side effects. Additionally, the nature of AAL systems demands that experts from different fields work together closely. Domain experts (like health care or nurture personnel) define appropriate use cases, and experts from different fields in computer science (e.g., pervasive computing to gain raw data, machine learning and pattern recognition to analyze these data, and human-computer interaction to allow implicit interaction with the system) implement the AAL system. These experts often make use of different programming languages and tools. As interoperability between different programming languages and tools is still a major problem in software development it might occur that programmers loose the general idea because their specific tools only provide a very limited view on the whole system. We believe that Model Driven Software Development (MDSD) is a suitable development paradigm that could solve these problems.

A main goal of MDSD in general and Model Driven Architecture (MDA) in particular is to separate the specification of the system functionality from the specification of the implementation of that functionality on a specific technology platform (see [13]). This seperation enables modeling an AAL system's complete functionality without the need to consider the restrictions of certain platforms. The platform-independent description (built, e.g., by domain experts) can then be transformed to the most suitable platform. MDSD tools generate code for specific platforms using code templates. Templates ensure a common reference architecture and quality standards for security and performance.

Furthermore, MDSD enables a prototyping-oriented development process. Many projects, as stated in [10], develop their applications in a bottom-up prototyping-oriented approach, in which few basic functions are quickly implemented and tested, and then incrementally refined. This is especially useful for machine learning applications because finding relevant features for optimal classification results requires tests with many different algorithm settings. But according to [15] a prototyping-oriented approach can only be successful if developers can build (potentially many) prototypes quickly and cheaply. This is not the case if prototypes are implemented without appropriate tool support.

In this paper we present how an MDSD prototyping tool can support domain experts and computer scientists in developing AAL systems. The MDSD prototyping tool can be used in a bottom-up modeling approach where domain experts use pre-defined components to build a system (as we described in our previous work, see [10]); additionally, we extended the tool with a top-down modeling approach, in which domain experts first model use cases, which might define placeholders for components that are still to be developed. These two approaches are comparable to the "includes" and "extends" concepts in UML use case diagrams.

The remainder of the paper is structured as follows: section 2 discusses related work, section 3 presents the MDSD concepts we used to implement the modeling tool, section 4 describes the development of a simple AAL system, and section 5 discusses the results and motivates further work.

2. Related Work

Model Driven Software Development (MDSD) is a software paradigm where software is (partly) generated from models (see [17]). Compared to MDSD in general Model Driven Architecture (MDA) is based on the Meta Object Facility (MOF) to ensure interoperability between models. Furthermore MDA emphasizes the separation between Platform Independent Model (PIM) and Platform Specific Model (PSM) as stated in [14]. The Eclipse Foundation, in particular the Eclipse Modeling Framework (EMF, see [3]) and the Graphical Modeling Framework (GMF, see [20]), provides rich support to simplify implementation of models and (graphical) editors enabling an MDSD approach.

The Context Recognition Network (CRN) Toolbox (see

[2]) describes a C++ framework integrating hardware abstraction, filter algorithms, feature extraction components and classifiers in a configurable runtime to support rapid development of context recognition applications. It is designed for deployment to embedded devices that support the POSIX runtime environment. Although a graphical editor is provided it is not based on a formal meta-model and thus the CRN Toolbox cannot fully benefit from an MDSD approach. As we base our toolbox on a formal meta-model we are able to generate code for various platforms, including the CRN Toolbox.

Dey et al. propose in [5] so called context widgets which acquire context information. Their main goal is to separate context information and the interpretation of these data from application logic by providing generic widgets (very much the same way as user interface widgets separate user input acquisition from data processing). Different sensors deliver data to widgets leading to a lot of communication effort beeing a critical issue in Wireless Sensor Networks (WSNs). Currently the widgets are only implemented in Java, thus they can not run on resource-constrained devices. Our approach provides platform independent and platform dependent components that are able to aggregate and interpret context information on different platforms (omitting communication overhead).

Lymberopoulos et al. (see [11], [12]) are using sensory grammars to model activities in assisted living. Their goal is to design and build a distributed asynchronous application—called BehaviorScope—using a large number of intelligent wireless sensors. High level semantics are derived from low-level sensor measurements using a hierarchy of sensory grammars. Their approach provides a structured way of interpreting macroscopic spatial and temporal activities similar to current automated speech recognition systems. A sensory grammars framework shifts the application development effort in programming the sensor network from low-level embedded programming to high level grammar scripting. The hierarchical modeling of activities can be compared to our approach which provides an abstract view on the system for domain experts. Different to our approach is that it depends on a framework, which is specifically tailored to the Crossbow sensor network platform. Moreover general data analysis components (like filters for feature extraction) and common machine learning algorithms are missing.

Tapia et al. showed in [19] a system for recognizing activities at home using a set of small and simple sensors. Preliminary results with state-change sensor installed in one-bedroom apartments showed that it is possible to recognize activities of interest to medical professionals such as toileting, bathing, and grooming. The major components of the systems are the environmental state-change sensors, a context-aware experience sampling tool, and pattern recognition and classification algorithm. The focus of this work was on recognizing activities with a wireless sensor network, but the authors did not address the software development issues.

TinyOS, see [7], is a component-based operating system for wireless sensor networks. It is widely used in developing applications for the popular Crossbow hardware. Applications for TinyOS are written in NesC (see [6]). Just recently Sentilla Java[1] emerged as operating system and programming framework for motes. These programming environments can all be seen as platforms in the sense of MDA.

3. Concepts and Tools for Model-Driven Development of AAL Systems

This section gives a short overview how the concepts of MDSD are applied for modeling AAL systems and which tools are used to implement these concepts (see figure 2). For further details please refer to [9] and [10].

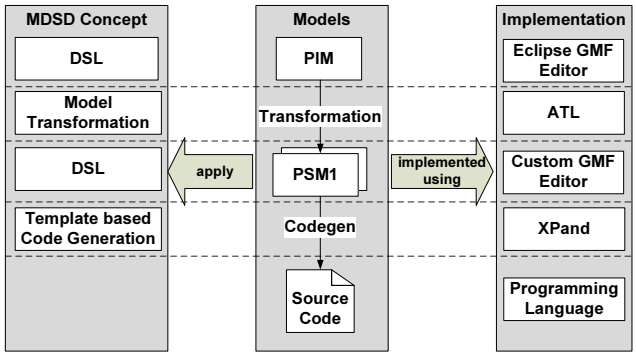

Figure 2. Concepts and tools applied for modeling AAL systems.

According to the MDA approach we capture the system functionality in platform independent models (PIM). The functionality specification is then refined with respect to a specific technology platform in so-called platform specific models (PSM). Thereby developers can reuse the PIM to implement a system on several platforms; additionally, they can specify complex applications in the PIM that can only be realized using several cooperating platforms. Domain specific languages (DSLs), which conform to corresponding meta-models, are used to describe AAL systems either in a platform independent or a platform-specific way. The platform independent meta-model defines, as introduced in [10]:

- *Readers* to interact with the system's environment and to deliver certain data, which we call *Aspects* (e.g., the vertical acceleration of an object)

[1] http://www.sentilla.com

- *Filters* to extract features from Aspects
- *Classifiers* to interpret Aspects and reason for new Aspects
- *Writers* to persist, communicate, or present Aspects to the user.

Furthermore we provide utility components like *Splitter* or *Merger* that can be used to split up aspects or to combine them. By applying a Pipes-and-Filters architectural style, see [4], these components can be combined to a system. *CompositeComponents* can be used to hierarchically group components, which enables stepwise refinement to manage complexity. They need not to be fully specified during modeling a PIM; optionally they can act as placeholders which are later specified in a separate platform independent model. A CompositeComponent's interface is defined with required and provided aspects.

A PSM combines the specification in the PIM with the details of a particular type of platform (see [14]). As we support different platforms it is possible to split up one PIM into several PSMs (see [10]). Every PSM again corresponds to a meta-model and thus defines its own DSL. To increase productivity we provide graphical editors (e.g., the PSM editor depicted in figure 4 on the next page) for our DSLs. The editors are based on the Eclipse Graphical Modeling Framework (GMF). Due to the fact that programmers should be able to easily integrate new platforms (and thus new PSMs) an abstract base editor for platform specific models provides feature rich base classes. By implementing extension points new components (e.g., a new filter) or new platforms can easily be integrated.

To simplify the transformation from a PIM to a PSM we provide a wizard that allows programmers to specify which component of the PIM should be transformed to which component of the PSM. Only transformations between components of the same type are allowed, e.g. a reader within the PIM cannot be transformed to a writer within the PSM. The transformations themselves are specified using the Atlas Transformation Language (ATL, see [8]).

After refining the details in the PSM executable code can be generated. A template-based code generation process using the XPand[2] template language is applied. The implementations of the components are provided by an archive file implemented in the platform-specific programming language, which is referenced by the generated code. Thus mainly glue code needs to be generated.

[2]http://www.eclipse.org/gmt/oaw/doc/4.1/r20xPandReference.pdf

4. A Top-Down and Bottom-Up Modeling Example

This section compares the top-down and bottom-up modeling approaches by means of an exemplary use case.

4.1. Use Case and Hardware Setup

A major goal of AAL systems is to increase or maintain a person's security or state of health by detecting deviations from normal behavior. Therefore we first need to learn a person's normal behavior (his or her daily routines) with machine learning algorithms (situation inference according to the reference architecture). The amount of raw data produced by sensors, however, is too large for this purpose; we need to recognize situations (i.e., interpret raw data), which can be further processed as features in higher-order situation inference engines.

The exemplary use case shows how to distinguish between two common household activities, which are still simple enough to be presented within the limited space of the paper: watching TV and vacuum-cleaning. These two activities are quite different but make use of common low-level components, which points out the advantages of the model driven software development process.

To distinguish between these situations we decided to use a Crossbow MICAz mote equipped with an MTS310 sensor board and TinyOS-2.x. The test person wears the mote at the ankle. We were interested in the acceleration values obtained from the sensor board's orthogonal accelerometers (we call these values x-acceleration and y-acceleration). The microphone of the MTS310 could be used to detect the specific sound of a vacuum cleaner. The next subsections discuss how this use case could be realized with the top-down approach and how this procedure differs from the bottom-up approach.

4.2. Top-Down Approach

In the top-down approach a domain expert models the system's functionality in terms of coarse-grained composite components (e.g. fall detection) which provide the needed results, and which might later be stepwise refined if necessary. Thus the model is an abstract view of the use case and is therefore modeled platform independently. The domain expert captures the user's requirements, preferences and constraints (e.g., a user may dislike image sensors but can imagine wearing a specific sensor, which might be integrated into the clothing). Additionally, the domain expert can observe characteristic behavioral patterns and environmental settings that indicate if a certain implementation is feasible or not.

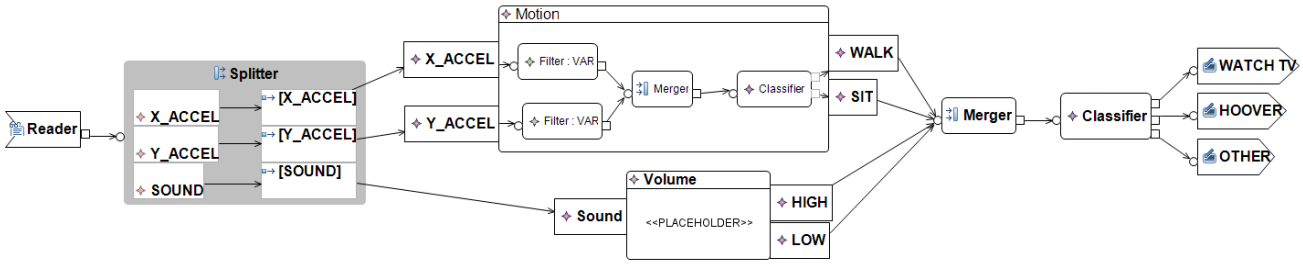

Figure 3. PIM modeled in top-down approach with placeholder and reusable composite component.

In our example the user is willing to wear an accelerometer and a microphone, which is captured by the reader's aspects in the model (see figure 3).

The application needs to distinguish between the household activities "watching TV", "vacuum-cleaning", and "other" (as defined by the classifier's output aspects). The domain expert's task is now to roughly describe the necessary steps to analyze the input aspects (using filters, classifiers, etc.) to produce the desired output aspects. In our case we get three different aspects which are split up in the first step. Next the question arises how to interpret the data to get the desired output. We assume that it is most likely that a person is sitting when watching TV and walking around when vacuum cleaning a room. When vacuum cleaning the volume level is significantly higher than when watching TV as the microphone is normally quite near to the vacuum cleaner. Thus a component is needed that is capable to decide if a person is walking or standing and another one that interprets the volume.

In the top-down approach these components are best defined using stepwise-refinement to manage modeling complexity. The domain expert has the choice to either select already pre-defined components, if they are available, or to define placeholders for such components which have to be refined by developers. The second choice adds the flexibility to configure the application with different implementations of one component specification; this is especially useful if the modeled application is designed for reuse in forthcoming use cases. The example contains both possibilities: the "Motion" composite component is pre-defined (as it might already have been used in a different use case), whereas the "Volume" composite component is a placeholder and can be configured later.

In the next step the abstract platform independent model is handed to the developer, who transforms it to the appropriate platforms. If the PIM contains placeholders, they first must be resolved either with an existing or newly created PIM. As described in section 3 the PIM can be transformed to several PSMs and finally source code is generated.

4.3. Bottom-Up Approach

In the bottom-up approach developers analyze the available hardware and its features. They typically start with simple platform-dependent prototypes which cover specific use cases. Figure 4 shows such a platform dependent prototype for the TinyOS platform (modeled in a TinyOS PSM). The prototype—a motion recognizer—decides whether a person is walking or sitting based on the accelerometer values' variance (which is discriminated with a threshold classifier to produce class labels). As the used MTS310 sensor board is able to deliver several aspects we decided to model only one reader. The aspects are split up using a splitter component. It would also be possible to model one reader per aspect and thereby omit the splitter component.

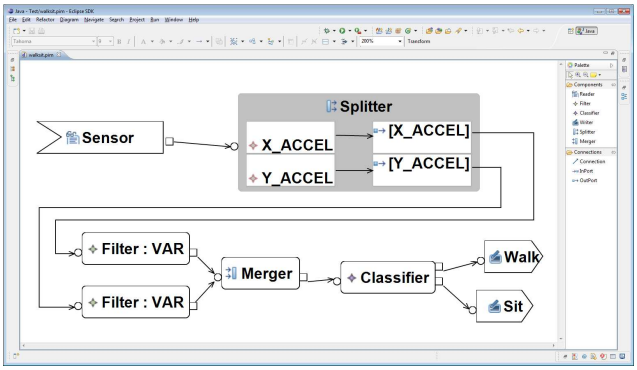

Figure 4. TinyOS PSM example motion recognizer.

To make such a prototype also available on other platforms developers can introduce a PIM, which describes the prototype independently from the platform. After the PIM has matured it could eventually be packaged into a reusable composite component that may be composed with other components to more complex applications. Concerning the specific example this could be done by removing the sensor reader and the splitter and define these aspects as required ones on the composite component. Similar the Walk and Sit

writer (which switch on different LEDs in TinyOS) must be modeled as provided aspects. This is where the top-down and bottom-up modeling approaches meet.

5. Conclusion and Further Work

In this paper we presented a new approach to build ambient assisted living systems. The intrinsic complexity of such systems, through different hardware platforms and programming languages, unreliable wireless communication, battery constraints, data analysis algorithms, recognition of situations and deployment options requires a specific development environment. This paper presents a development environment which supports domain experts in their top-down approach, as well as technically experienced developers in their bottom-up approach. The implemented tools facilitate model-driven software development from platform-independent modeling to generating AAL application code. The implementation of several use cases showed promising results, but the development environment must now be further evaluated to measure its advantages and shortcomings. We currently do this especially for health care and eldercare.

Acknowledgment This research was supported by the Upper Austrian state government. Any opinions, findings, and conclusions or recommendations in this paper are those of the authors and do not necessarily represent the views of the research sponsors.

References

[1] H. Ailisto, P. Alahuhta, V. Haataja, V. Kyllönen, and M. Lindholm. Structuring Context Aware Applications: Five-Layer Model and Example Case. In *Proceedings of Ubicomp 2002, Concepts and Models for Ubiqutious Computing Workshop*, 2002.

[2] D. Bannach, O. Amft, and P. Lukowicz. Rapid prototyping of activity recognition applications. *IEEE Pervasive Computing*, 7(2):22–31, April–June 2008.

[3] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. Grose. *Eclipse Modeling Framework*. Addison Wesley, 2004.

[4] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture 4: A Pattern Language for Distributed Computing*. John Wiley and Sons Inc., 2007.

[5] A. K. Dey, D. Salber, and G. D. Abowd. A context-based infrastructure for smart environments. In *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99)*, pages 114–128, 1999.

[6] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the International Conference on Programming Language Design and Implementation*, San Diego, CA, USA, 2003. SIGPLAN.

[7] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, USA, 2000. ACM.

[8] F. Jouault and I. Kurtev. Transforming models with atl. In *Proceedings of the Satellite Events at the MoDELS 2005 Conference*, volume 3844/2006, pages 128–138. Springer Verlag, 2006.

[9] W. Kurschl, S. Mitsch, and J. Schoenboeck. An engineering toolbox to build situation aware ambient assisted living systems. In *Proceedings of 3rd International Conference on Broadband Communications, Information Technology and Biomedical Applications*, Pretoria, South Africa, November 2008. IEEE.

[10] W. Kurschl, S. Mitsch, and J. Schoenboeck. Model-driven prototyping support for pervasive health care applications. In *Proceedings of the 9th IASTED International Conference on Software Engineering and Applications*, Orlando, Florida, November 2008. ACTA Press.

[11] D. Lymberopoulos, T. Teixeira, and A. Savvides. Detecting patterns for assisted living using sensor networks: A case study. In *Proc. International Conference on Sensor Technologies and Applications SensorComm 2007*, pages 590–596, 14–20 Oct. 2007.

[12] D. Lymberopoulos, T. Teixeira, and A. Savvides. Macroscopic human behavior interpretation using distributed imager and other sensors. *Proceedings of the IEEE*, 96(10):1657–1677, Oct. 2008.

[13] J. Miller and J. Mukerji. Model driven architecture (mda). http://www.omg.org/docs/ormsc/01-06-01.pdf, 2001. last accessed 2008-09-10.

[14] J. Miller and J. Mukerji. Mda guide version 1.0.1. http://www.omg.org/docs/omg/03-06-01.pdf, 2003. last accessed 2008-09-11.

[15] G. Pomberger and R. Weinreich. The role of prototyping in software development. In B. Magnusson, B. Meyer, J.-M. Nerson, and J.-F. Perrot, editors, *Proceedings of the 13th Intl. Conference on Technology of Object-Oriented Languages and Systems*, Versailles, France, 1994. Prentice Hall.

[16] K. Roemer. *Time Synchronization and Localization in Sensor Networks*. PhD thesis, Swiss Federal Institute of Technology Zuerich (ETH Zuerich), 2005.

[17] T. Stahl and M. Voelter. *Modellgetriebene Softwareentwicklung*. dpunkt.verlag, 2005.

[18] H. Steg, H. Strese, C. Loroff, J. Hull, and S. Schmidt. Europe is facing a demographic challenge - ambient assisted living offers solutions. Technical report, VDI/VDE-IT, 2006.

[19] E. M. Tapia, S. S. Intille, and K. Larson. Activity recognition in the home setting using simple and ubiquitous sensors. In A. Ferscha and F. Mattern, editors, *Proceedings ot PERVASIVE 2004*, volume LNCS 3001, pages 158–175. Springer Verlag, 2004.

[20] M. Voelter, A. Haase, S. Efftinge, and B. Kolb. Graphical modeling framework. *iX*, 12:17, 2006.

[21] D. Zhang, T. Gu, and X. Wang. Enabling context-aware smart home with semantic web technologies. *Intl. Journal of Human-friendly Welfare Robotic Systems*, 6:9, 2005.