A Reusable Software Architecture for Context-Aware Applications

Feng-Pu Yang, Fang-Chen Hwang, Wei Zhang Hu, Yu-Sheng Lai,
Hewijin Christine Jiau and Pau-Choo Chung
National Cheng Kung University Electrical Engineering,
Industrial Technology Research Institute
e-mail: jopwil@nature.ee.ncku.edu.tw

Abstract

As the popularity of ubiquitous computing and pervasive computing, the goal of building smart home applications becomes practical. Context-awareness is a key feature of good smart home applications, but many context-aware features have been hard-coded into traditional applications. The hard-coded features cause the slow development of smart home application, and also let the smart home application become inflexible to varying when the requirement is changed. This paper aims to improve the productivity of the development of smart home application by increase the possibility of both reuse and adaptation. By analyzing the variations occurred within smart home domain, we make those possible variations as configurable tuning points. Those variations can be adapted through configuring tuning points, and there is no need to modify the program frequently when the situation is changed.

Keyword：context-aware, smart home, configuration, software architecture

1. Introduction

The popularity of ubiquitous computing [10], where computer moves off the desktop and into the environment, and pervasive computing [4], which provides anytime and anywhere computing by decoupling user from device, have prompted the proliferation of context-aware applications [7]. Such context-aware application, which can adapt to changes in the environment and human's activities dynamically, can be used in various application domains, such as smart homes. A smart home [5] uses networked sensors, devices, and appliances to build an intelligent environment in which many features in the home are automated and where devices and services seamlessly cooperate to support household tasks. An envisioned scenario of smart home application would be like that: Grandmother wakes up in the middle night, and she goes through hallway to bathroom. The system senses that the hallway is dark, and turns on the dim light instead of dazzling light because the time context is midnight. As the scenario mentioned above, smart home system can provide proper and customized services for each home member automatically by the using of context-awareness. However, those context-aware features have been hard-coded into traditional context-aware applications, which make development slow and inflexible to varying when the requirement is changed. This paper aims to speed the development process by enhancing the reusability and also provide flexibility against the requirement changes within smart homes application domain.

2. Catalog of Variations

As mentioned above, context-aware features have been hard-coded into traditional context-aware applications. Many researchers dealt with this problem by introducing configuration into their design. However, their configurations are still entangled with their usage, and developers who want to configure such a system need to dig into the codes. For example, the work of S. Helal et al. [6] is the typical paper addressed the hard-coded problems by inventing an architecture, which can help context-aware application developers to build their applications on top of existed applications. This approach increases the productivity of individual developers through a kind of reuse, but it still ignored a serious problem of adaptability. The power of this architecture is to accumulate the reusable context-aware applications for further developers to develop with reuse. The bigger the number of accumulated applications causes the higher the possibility of reuse. However, the power will reduce dramatically when the system needed to be adapted to other environments. Because the architecture of [6] did not take the varieties of environments and home members in advance, the developed applications entangle the functionalities with a specific setting of environment and home member. People who needed to adapt this software to a new setting of environment and home member should pay considerable effort to identify the related portions and then modify them. In other words, the reusability obtained under such kinds of architecture is limited within the boundary of a specifically predefined environment setting. When the boundary is shifted, the architecture can not promise the reusability any more. To preserve the benefits of accumulated reusability across the boundary to some extend, researchers need to consider more on possible variations carefully. Current researches [8] [5] [6] [1] mainly focus on the extensibility issues, but they are unconscious of the adaptability issues. From the perspective of maintenance staffs, the current researchers and this paper both focus on preventive maintenance. The difference is that current researchers focus only on the aspect of easing effort of perfective maintenance in the future and this paper focus on the aspect of easing both the effort of perfective and adaptive maintenance in the future [13].

In order to deal with such problems, the main principle of this paper is to separate configuration from use. Therefore, we need to identify the variations within smart home domain, and then extract them outside to serve as tuning points. By separating these tuning points outside, the developed system is able to generalize to other settings with much fewer efforts then before.

After the analyzing a series of smart-home scenarios, we introduce two more variations, environments and home members, in addition to the sensors and services. Sensors and services can not be linked directly without taking environments and home members into consideration. Those variations will be discussed through section 2.1 to section 2.4.

2.1. Variations of Sensors

Location is an important context in almost all context-aware applications, and there are much more than one kind of sensors can provide context of location. Active badge [2] is a good candidate of location-related sensing. Each active badge is associated with a unique identity, which is broadcast as part of the beacons emitted by the badge worn by a home member. The badge infrastructure consists of a group of badge readers that are deployed in various locations in the smart home environment. However, the level of resolution may not be appropriate for many

applications, which require even finer granularity. Such kind of applications may need some other sensors such as supersonic sensors. No matter what sensors are in use, almost every smart home application needs a location sensor, but the interpretation mechanisms of those location sensors are hard-coded into code in traditional context-aware applications. That causes two serious problems. First, such kinds of design force developers to recreate solutions to problems that are already solved. Because the sensor interpretations are hard-coded with what the applications want to do. The effort to refactor such code to reusable code may be more than the effort to write from scratch. Second, such kinds of applications need to be reprogrammed when the location sensors are changes, such as changing from active badge to supersonic sensor.

2.2. Variations of Environments

To deal with the variation of environment, developers need to carry out the adaptive maintenance [13], which means software maintenance is performed to make a computer program usable in a changed environment. A smart home environment's variations come from the scale of house, the internal design of house, the deployment of sensors and the cultures of using each zone of a house. An extreme example of scale variation is the house of Bill Gate. For example, Bill Gate may need lots of location sensors cooperate to get overall information of his living room under such a large scale. Internal design is another factor, which may be correlated with the cultures of using each zone of a house. These two kinds of variations affect the treatment of sensor data and the style of services provision. For example, the locations of sensors are strongly influenced by the home environment in both positive and negative ways. Some internal designs may limit the available spaces for sensor deployment, and some designs will be suitable for deploying specific kind of sensors. All the sensor deployments with respect to scales, the internal design and culture of usage form the environment of smart home application. One of this paper's goals is to prevent those environment variations being hard-coded into smart home application. The other goal is to reduce the effort of adaptive maintenance when the home environment is change through enhancing reusability.

2.3. Variations of Home Members

The variations between home members cause the interpretation of the same context becomes differently. For example, the same temperature may have different interpretations for adults and elders. Take the variations of home members into consideration will let the behavior of system become more adaptive to each home member. In addition to provide adaptive services to each home member, another important concern is to keep home member with minimal disturbance when system provides services. Traditional applications usually deal with this problem by introducing user profiles [4] [9] [13]. This approach has been proven to be practical, and we use this approach to record the variations of home members for adapting to each home member and minimizing disturbance.

2.4. Variations of Services

The types of services provided by service providers would be influenced by many factors, such as demands of their target users. Because different human in different environments will have different needs, and the corresponding services are sought to deal with different situations. If developers do not take the variations of services into account, they will

pay lots of effort when the demand is changed. For example, they may need to redeploy sensors, to write the suitable program to interpret the sensors' data and to link those interpreted data with newly deployed actuators. In order to prevent deploying duplicated sensors and programming duplicated code, we identify some general commonalities and variations of information that services needed under the domain of smart home. Those identified information can be viewed as the guidance for the context-aware engine to generate the useful information. The service providers can find mostly all information they needed from information generated by guidelines.

3. Separation of Concern

In addition to separate configurations from use, this paper aims to increase the reusability of context-aware system. This section introduces the approach of reusability enhancing by separation of concern between variations.

There are some dependencies between the variations mentioned in section 2. For example, auto-control services of air conditioner usually can not perform without the assistance of thermometer. That is an example of dependency between sensors and services (the available sensors will affect what services the system can provide to some extend). Dependency between sensor and environment is like that the interpretation of the location sensors of a villa may be different with the location sensors of an apartment. If we just put all of those variations into one configuration file in a flatten structure, the modification of one portion of configuration will affect other portions. This phenomena is like the change propagate problem in software maintenance, and we need to reorganize the configuration to make sure the change will only happen locally. By analyzing the dependencies between those variations, a hierarchic structure is built to manage and organize the variations. Figure 1 describes the separation of concerns; **Control Parameter** deals with the variations of sensors, **Composite Constrain** deals with the variations of environment and **Core Ontology** deals with the variations of services as well as **User Profile** which deals with the variations of home member.

3.1. Primitive Context and Control Parameter

The context comes directly from sensors with proper interpretation specified in the **Control Parameter** is called **Primitive Context**. **Control Parameter** records the interpretation between sensor data and **Primitive Contexts**. **Primitive Context** encompasses the information of sensed type, sensor ID, sensor location and sensor data. The sensed data is the interpreted result of sensor's raw data, and the interpretation rules are defined in the **Control Parameter**. For example, the thermometer senses the temperature of 80℃ may be interpreted as hot because the **Control Parameter** defines temperature which is higher than 50℃ as hot. **Control Parameter** reduces the burden of the following processing by giving basic meaning of raw data. In addition to reduce the stream of data provided by the sensor, **Control Parameter** also helps the further stages to determine which **Primitive Contexts** are relevant based on those basic meaning. The variations of sensor are taken into consideration in **Control Parameter**.

3.2. Composite Context, Composite Constrain and User Profile

The context, which is composed of two or more **Primitive Contexts** under constrains specified by **Composite Constrain** is called **Composite Context**. The ability of grouping relevant stimulus and ignoring irrelevant stimulus is important for human to be

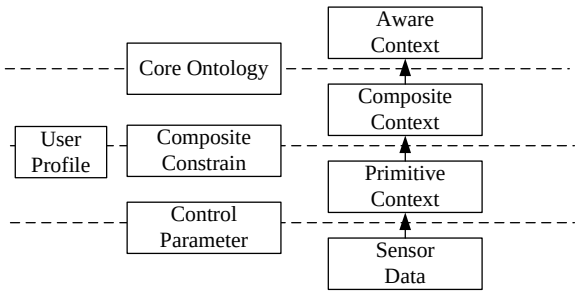

Figure 1: Conceptual Model of context-awareness applications.

aware of environment changes. If we want computers to cope with our daily life, the first thing we should do is to make the computer be able to group meaningful stimulus. There is a need to group relevant **Primitive Contexts** by some properties such as time, locations, and home member ID, and it is often the case that a service requires multiple **Primitive Contexts** which possessed a common value of a specific property. For example, the **Primitive Context** of "Home member #1 enters living room" and the **Primitive Context** of "the light of living room is off" can be combined as a **Composite Context** of "Home member #1 enters living room where the light is off", and such a combined context may be useful for light control services. Living room is the common property value for these two **Primitive Contexts** to be combined.

The **Composite Constrain** records constrains of how to group relevant **Primitive Contexts** and ignore irrelevant **Primitive Contexts**. **Composite Constrain** is defined on grouping relevant **Primitive Contexts** such as time, location and home members ID. **Composite Constrain** contains the information of time, involved home member, and location, which are related to the grouping. The variations of environment are taken into consideration in **Composite Constrain**.

To generate **Composite Contexts** needs the supporting information of home members. In order to take the variation of home members into concern, the user profile is used for providing information of each home member. This approach will let the behavior of system become more adaptive to each home member. In addition to provide adaptive services to each home member, the other advantage of using user profile is to keep home member with minimal disturbance when the system provides services. For example, it is not comfortable if home members need to answer a lot of questions, such as the preference of temperature and brightness, before entering their bedroom.

Current applications of user profile [4] [13] [9] can be divided into two types. Type I records configuration settings and other data associated with a user, and type II records demographic information and statistics about that user's behavior. The purpose of using type I user profile is to prevent the user being aware of environment change by providing location transparency. However, that is not suitable to the context-aware application, which needs to be aware of environment changes. Type II user profile is more suitable for the purpose of context-aware applications. The variations of home members are taken into consideration in **Composite Constrain** with the assistance of user profile. The user profile used by this paper belongs to Type II.

3.3. Aware Context and Core Ontology

Aware Contexts are the inferred contexts, which occur when some composite contexts are in the same conditions specified by **Core Ontology**. For example, the **Aware Context** "Housebreaking when the occupants are absent" is occurred when the **Composite Contexts** "No home members at home" and "Housebreaking" coexisted concurrently. The conditions of **Aware Contexts** occurrence are specified in **Core Ontology**.

Human can not experience all things in the

world, but he can learn from experience. By sharing self-experience with each other, human learn things more efficiently. Sensor-intensive approach gives computer the capability to experience the world. That means it is possible for a computer to learn something from its surround environment. However, it is time-consuming to let computer learn by itself from scratch. In order to make computer become ready for use as soon as possible, we build a **Core Ontology** to accumulate experience of human beings for computers. Human can share their experiences with computers through using **Core Ontology**, and computers which are aware of **Core Ontology** can help human in many different aspects in a human-liked way.

Core Ontology can be extended through updating or being added locally. The organization which is responsible for the **Core Ontology** management should provide the services of **Core Ontology** updating, **Core Ontology** synchronization…etc.

4. System Architecture

As the Figure 2 depicts, the whole system is composed of four functional modules, which are **Context Acquisition** module, **Context Aggregation** module, **Inference Engine** module and **Aware Contexts and Service Mapping** module. Two Databases, **Current Environment DB** (database) and **Core Ontology DB,** are used to support functional modules. The responsibility of **Context Acquisition** module is to transform raw data to **Primitive Context**s. Those **Primitive Context**s generated by **Context Acquisition** module are combined by **Context Aggregation** module to generate meaningful **Composite Contexts**. **Inference Engine** receives those generated **Composite Context** and then deduces more information, which called **Aware Context**, out of them. Finally, **Aware Contexts and Service Mapping** module maps interested **Aware Context** to services by the demands of service providers. In this section, the main functional modules are explained sequentially below.

4.1. Context Acquisition

Sensor sends the measured or detected data to the **Context Acquisition** module. **Context Acquisition** module interprets the receiving data by refer-

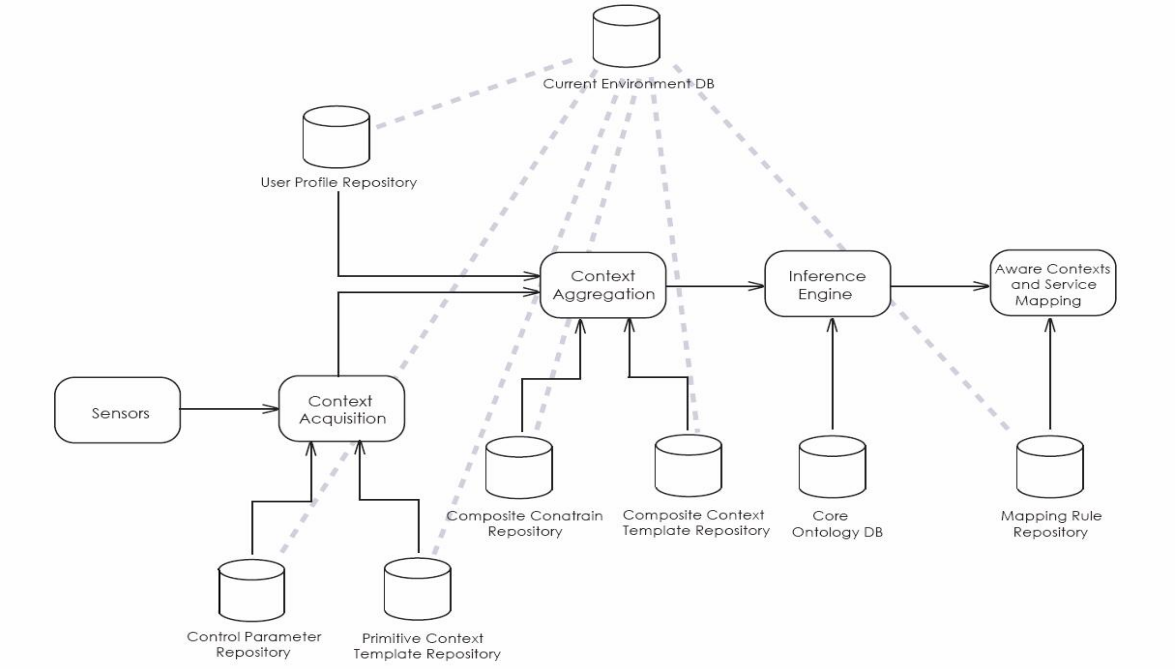

Figure 2: System architecture

encing the corresponding **Control Parameter** from the **Control Parameter Repository**. The interpreted data are filled in the corresponding template, which extracted from the **Primitive Context Template Repository**, and sent as a **Primitive Context** to **Context Aggregation** module for further processing.

Context Acquisition module provides a separation of concerns between how **Primitive Context** is acquired and how it is used by **Context Aggregation** module. The domain experts can use **Control Parameters** to specify the transformations between sensor data and **Primitive Contexts**. When the system is migrated to a new environment, some **Control Parameters** can be reused with, if any, few modifications when the new environment has similar sensors in use.

4.2. Context Aggregation

Context Acquisition module sends the **Primitive Context**s to the **Context Aggregation** module. **Context Aggregation** module groups the arrived **Primitive Context**s by time, location, home member ID and other grouping properties with the assistance of underlying grouping mechanisms. **Context Aggregation** module checks the occurrence of meaningful combinations within current **Primitive Context** set by referring to **Composite Constrain**, which is loaded from **Composite Constrain Repository**. The generated meaningful combinations are filled in the corresponding template, which extracted from the **Composite Context Template Repository**, and sent as a **Composite Context** to **Inference Engine** for further processing.

In order to support the grouping operation on such properties, there is a need to design corresponding mechanism for each operation. Those corresponding mechanism are encapsulated in the form of components, which are deployed in **Context Aggregation** module to support new grouping operations. A. K. Dey [1] had mentioned that with no support for aggregation, an application has to use a combination of subscriptions and queries on different **Primitive Context**s to determine when these conditions are met. It is unnecessarily complex and is difficult to modify if changes occur.

4.3. Inference Engine

Inference Engine receives **Composite Contexts** from **Context Aggregation** module. The **Core Ontology** is loaded from **Core Ontology DB**. When those incoming **Composite Contexts** match the fire condition defined in the **Core Ontology**, the corresponding **Aware Context** will be fired. **Aware Contexts** are sent to the **Aware Context and Service Mapping** module for further processing.

Inference Engine is responsible to interpret the current situations in terms of high-level concepts, which are understandable by human. The **Core Ontology** consists of many **Composite Context** nodes and relationships between those **Composite Context** nodes as well as firing conditions of **Aware Contexts**. The composers of **Core Ontology** need not to specify all the relationships between **Composite Context** nodes explicitly, the **Inference Engine** will infer the correct relationships automatically. The inferred **Core Ontology** is then used for runtime operation. Each incoming **Composite Context** is classified to corresponding **Composite Context** node of **Core Ontology** instance, and the relationships between **Composite Context** nodes are monitored by the **Inference Engine**. When the occurred relationships match any fire condition, the corresponding **Aware Context** will be fired.

4.4. Aware Context and Services Mapping

Inference Engine generates **Aware Contexts** to **Aware Context and Service Mapping** module. The **Aware Context and Service Mapping** module maps each **Aware Context** to the services which need the aware context by referring the **Mapping Rule** stored in **Mapping Rule Repository**.

Inference Engine exposes all the possible **Aware Contexts** for services' usage. The service providers can link their services to corresponding **Aware Contexts** by composing their **Mapping Rules**. When the service providers want to provide new services, what all necessary they should do is to modify the **Mapping Rules**.

5. Related Works

Researches [8] [5] [6] [1] are also aware of the importance to deal with some variations which mentioned in section 2. For example, A. Schmidt et al. have used cue to deal with the variations of sensors [8]. The concept of cues provides an abstraction of sensors. Each cue is dependent on single sensor; but using the data of one sensor, multiple cues can be calculated. When including new sensors with different characteristics, only changes in the corresponding cues must be adapted, cues are also a way to reduce the stream of data provided by the sensor. The role of cue is similar as the **Control Parameter** in this paper, but A. Schmidt et al. did not take other variations into considerations. T. Gu et al. proposed a hierarchical context ontology [5], which separates the domain specific ontology from the common high level ontology. Instead of dealing with domain, this paper deals with environment directly. The reason of such a design decision is that the possible application domain of a smart place is not usually predictable, but the application domain must depend on its environment to some extend. T. Gu et al. did not take sensor variations into concern, and what they have mentioned is the platform independent because of using Java. Developers who want to apply the architecture should take the variations of sensors into consideration. The Gator Tech Smart House [6] focus on creating a smart apace which has the ability to evolve as new technologies emerge or as an application domain mature. They have wrapped the existed sensors to incorporating some new sensor technologies, which called smart sensors. In comparison with this paper, the Gator Tech Smart House also extracts sensor variations and wrapped those variations in the form of OSGi service. Such approach can prompt reusability as what this paper prompted, but the approach can not provide flexibility as this paper provided. A. K. Dey proposed an architecture support [1] to deal the variations of sensors. A. K. Dey provided many context toolkits for building context application, which can fulfill the first goal of this paper to build context-aware application with high productivity. However, the learning curve of context toolkit is higher than the learning curve of tuning points mentioned in this paper. As the context toolkits framework becomes larger and larger, the learning effort would increase dramatically. The advantage of this paper in comparison with the related works is that this paper deals with those variations explicitly. By a systemically analysis of each variation, the design can survive well against the changes of those variations. Separating those variations from common features can also prompt design with reuse.

6. Conclusion and Future Work

The variations of services, environments and members as well as sensors are analyzed for separating the configuration from use. Because the variations are not orthogonal, the conceptual model is designed in a hierarchical style based on the dependencies between those variations. Software architecture

is presented to realize the conceptual model of this hierarchical design, which has three levels of context processing. The three levels of context processing are context acquisition, context aggregation and inference engine, and their behaviors can be tuned by configuring corresponding tuning points, which are **Control Parameter**, **Composite Constrain** and **Core Ontology**. The main purpose of **Control Parameter** is to deal with the variation of sensors and to separate the use of **Primitive Context** from the detail of **Primitive Context** acquisition. **Composite Constrain**s is used to aggregate meaningful context combinations and to filter out irrelevant context combinations. **Core Ontology** is a record about human experiences of smart home environment, and it can be accumulated to be a knowledge-rich database which can be used across many domains in the future. In addition to those tuning points mentioned above, the variations of home members are also taken into account through the use of user profile. By separating variations into configurable tuning points, the approach of this paper increases the flexibility. This paper also enhances reusability by introducing hierarchical modulation of tuning points. Next step will be deploying such a context-aware smart home system in physical houses, and carry out some experiments for evaluating the degree of improvement in flexibility and reusability.

7. Acknowledgments

This paper is a partial result of Project B34BSP1300 conducted by ITRI under sponsorship of the Ministry of Economic Affairs, R.O.C.

8. Reference

[1] A. K. Dey, "Providing Architectural Support for Building Context-Aware Applications," *PhD thesis, College of Computing, Georgia Institute of Technology*, Dec. 2000.

[2] D. J. Cook and S. K. Das, "Smart Environments: Technology, Protocols and Applications," *Wiley-Interscience*, Oct. 2004.

[3] D. Garlan, D. Siewiorek, A. Smailagic and P. Steenkiste, "Project Aura: toward Distraction-free Pervasive Computing," *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 22-31, April-June 2002.

[4] G. P. Eleftheriadis and M. E. Theologou, "User profile identification in future mobile telecommunications systems," *IEEE Network*, vol. 8, no. 5, pp. 33 -39, Sept.-Oct. 1994.

[5] T. Gu, H. K. Pung and D. Q. Zhang, "Toward an OSGi-based Infrastructure for Context-Aware Applications," *IEEE Pervasive Computing*, vol. 3, no. 4, pp. 66-74, Oct.-Dec. 2004.

[6] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura and E. Jansen, "The Gator Tech Smart House: a Programmable Pervasive Space," *IEEE Computer*, vol. 38, no. 3, pp. 50-60, March 2005.

[7] Schilit, Bill N., Norman I. Adams and Roy Want (1994). "Context-Aware Computing Applications," *1st International Workshop on Mobi-Comp. Systems and Applications*, pp. 85-90 Dec. 1994.

[8] A. Schmidt and K. van Laerhoven, "How to Build Smart Appliances?," *IEEE Personal Communications*, vol. 8, no. 4, pp. 66-71, Aug. 2001.

[9] S. E. Middleton, N. R. Shadbolt, and D. C. De Roure, "Ontological User Profiling in Recommender Systems," *ACM Trans. on Information Systems*, vol. 22, no. 1, pp. 54-88, Jan. 2004.

[10] M. Weiser, "The Computer for the 21th Century", *Scientific American*, vol. 265, no. 3, pp. 94-101, Sept. 1991

[11] U. Hansmann, L. Merk, M. Nicklous, T. Stober, "Pervasive Computing Handbook, " *Springer-Verlag*, 2001

[12] Y. Zhiwen and Z. Xingshe, "TV3P: An Adaptive Assistant for Personalized TV," *IEEE Trans. on Consumer Electronics*, vol. 50, no. 1, pp. 393-399, Feb. 2004.

[13] "IEEE Std. 1219-1998: IEEE standard for software maintenance," *IEEE Std*, Oct. 1998.