# SOFTWARE ARCHITECTURE FOR COOPERATIVE BUILDINGS

**Dr. Klaus Bergner, Andreas Rausch, Marc Sihling, Alexander Vilbig**
Chair for Software & Systems Engineering, Department of Computer Science
bergner@in.tum.de, rausch@in.tum.de, sihling@in.tum.de, vilbig@in.tum.de

**Katja Popp**
Chair for Building Realization and Computer Science, Department of Architecture
katja.popp@bri.arch.tu-muenchen.de

**Thomas Reicher**
Chair for Applied Software Engineering, Department of Computer Science
reicher@in.tum.de

***Technische Universität München, Arcisstraße 21, 80333 Munich, Germany***

In this paper, we propose an overall software architecture for the application domain of cooperative buildings. Based on an analysis of the predominant functional requirements, we develop a business-oriented conceptual framework, establishing a common understanding of a cooperative building's entities and their relations. The framework is then mapped to an abstract technical software architecture with suitable system components and interactions. The resulting overall architecture provides a unifying high-level model of the various existing technical infrastructures for cooperative buildings, upon which interoperable solutions may be constructed.

**Keywords:** cooperative buildings, software architecture, business-oriented framework, componentware

## 1 INTRODUCTION

For years, market analysts have predicted substantial growth in the area of building automation, as all kinds of buildings and their inhabitants may benefit from advances in information technology. Integrated communication and monitoring facilities for buildings and so-called "intelligent" devices may raise the standard of living for end users, for example, by adjusting automatically to the preferences of their users. Furthermore, they open up new possibilities for optimizing aspects like maintenance costs and energy consumption.

However, existing approaches for cooperative buildings have not been very successful yet. Although individual devices as well as complete home automation systems are already available on the market, they are not employed by most people due to various restrictions. Important drawbacks are the incompatibility between products of different manufacturers, the need for skilled experts to install or even use them, or the costs associated with purchase and maintenance. Clearly, there is a need to develop standardized, interoperable, easily usable, and considerably cheaper products in the area of building automation.

A comprehensive software architecture represents a key factor for the successful integration and standardization of information and building technology. It implies a level of abstraction well above technical details of individual communication networks and mechanisms, thus enabling the design of general, interoperable mechanisms based on reusing existing components. The software architecture proposed in this paper consists of two main parts: an abstract business-oriented model and a technical software architecture.

The business-oriented model presented in Section 2 is not tied to a certain technical infrastructure. Instead, it may be mapped onto different implementation platforms, as long as the resulting technical architecture fulfils the given non-functional requirements. This separation of business-oriented issues from technical issues raises the clarity of the design and allows to reuse the business model even if the underlying technology changes [BRSV98]. In particular, it captures the core concepts of the cooperative building domain, like 'device', 'location', or 'service', and allows to model the relations between these entities at an abstract level.

Section 3 presents a technical software architecture that employs the proposed business-oriented model to describe and realize system functionality offered as services to the inhabitants of a building. It provides a decomposition of the system into components and defines the necessary interaction protocols, for example when installing a new device in the building. We propose a layered organization of the architecture with clear abstractions and interfaces between application level, framework services, and communication infrastructure. This kind of architecture makes it possible to exchange technical solutions on each individual level, thus leveraging the benefits of heterogeneous, vendor-independent systems.

Finally, a short conclusion summarizes the main results of the paper.

## 2 FUNCTIONAL REQUIREMENTS AND BUSINESS-ORIENTED MODEL

In this section, we first identify the essential functional requirements for systems within a cooperative building. Based on this, we present an abstract business model of the corresponding system entities.

### *2.1 Functional Requirements*

The following abstract functional requirements characterize the application domain as they are crucial for gaining user acceptance:

*Ease of Use:* This is the predominant requirement for most conceivable use cases of a cooperative building. Installation, configuration, usage, and de-installation of cooperative devices have to be as simple as possible.

*Security:* As home appliances are directly or indirectly used by different groups of people, it is highly important that the system offers provisions for guaranteeing the privacy of data and for preventing unauthorized access to services.

*Awareness of Space:* The system must be able to deal with spatially distributed users and devices. First, locations should be modeled explicitly to allow devices to query their current position and to adjust their behavior accordingly. Second, the system should be able to cope with mobile users and devices.

*Awareness of Time:* Time constraints need to be explicitly captured within the system. This is crucial, for example, with regard to the time between raising an alarm and consecutive reactions of the user.

### *2.2 Business-Oriented Model*

The proposed business-oriented model abstracts away from the concrete characteristics of special devices, systems, and services. From such an abstract viewpoint, a typical system configuration of an apartment may look like visualized in Figure 1.

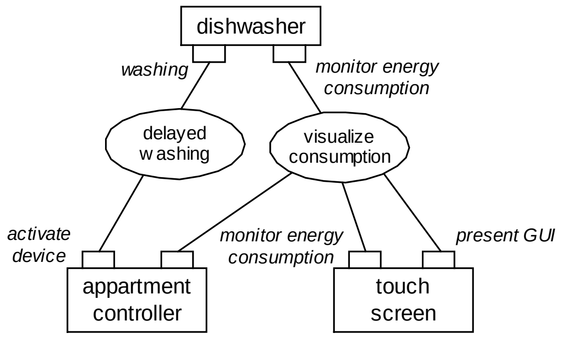

Figure 1: Exemplary System Configuration

The figure shows three devices in an apartment, namely, a dishwasher, a central controller device representing the apartment, and a touch screen. Each device, visualized by a large rectangle, has different characteristics and capabilities, visualized by smaller, adjacent rectangles. The dishwasher, for example, is able to wash the dishes and to monitor its energy consumption. Currently, the visualized devices cooperate in order to perform two services: The apartment controller has activated the dishwasher (based on a special tariff scheme provided by the local energy provider), and the touch screen shows the combined apartment consumption resulting from the activity of the three devices.

Figure 2 provides an abstract, business-oriented model capturing the essential concepts as discussed above. The model is presented as a UML class diagram in order to facilitate its translation to an object-oriented framework.

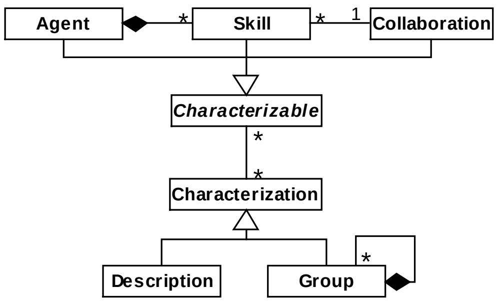

Figure 2: Abstract Business-Oriented Model

In the following, we describe the two main layers of the model, namely, the instance level in the upper part, and the characterization level in the lower part.

*Instance Layer:* The instance level consists of the three classes Agent, Skill, and Collaboration, and the interface Characterizable.

*Agents* model actors like devices or user agents. Agents cooperate with other agents in order to achieve a certain goal.

*Skills* model the capabilities of agents. The dishwasher, for example, offers the “washing” and “monitor energy consumption” skills. Other agents may rely on these skills in order to perform their duty. Each agent is fully responsible for managing its skills. Agents are never accessed directly, but exclusively via their skills. As dedicated access points to their agent, a skill may also imply certain safety and security constraints.

*Collaborations* model the cooperation and the admissible communication protocols that must be observed by the participating agents in communicating via their skills. A collaboration involves a number of agents which contribute their respective skills in order to render a certain service to the user.

Note that the possible connection structures which may be built with these three concepts are highly dynamic and cannot be restricted a priori. Usually, the agents will have the responsibility to offer their skills to other agents and to establish collaborations.

Profound knowledge about the properties and characteristics of agents, skills, and collaborations is necessary in order to establish collaborations. Therefore, we introduced a general interface “Characterizable”, which is implemented by corresponding classes. Characterizable entities are denoted by a so-called “Characterization”, a general concept for specifying arbitrary sets of instances. Note that characterizations capture not only instances of a certain kind (for instance, only agents), but also mixed sets of instances (for instance, some agents and their skills). An instance may belong to multiple characterizations at the same time. Our conceptual framework contains two standard subclasses of “Characterization”:

*Descriptions* capture static instance properties that remain stable over long periods of time. Examples for descriptions are object-oriented signatures or graphical interaction descriptions like sequence diagrams which may be modeled as subclasses of Description. Note that this provides a framework for adding various kinds of description techniques as well as the corresponding relationships between them in the future. In the example of Figure 1, descriptions supplied by the device vendors could determine the protocol necessary for querying energy consumption data from a device as well as the admissible connection structure between the devices.

*Groups* capture volatile instance properties that may change multiple times during operation. A good example for a subclass of Group is the Location class, modeling real-world locations like rooms in cooperative buildings or streets in a town. Groups may form hierarchies, like the spatial containment hierarchy of locations.

Descriptions and groups are base concepts for both device vendors and users. Standard descriptions provided by vendors enable users and other devices to query the capabilities of new devices and to use them in a flexible way. More complex descriptions may be useful as a specification for component vendors or even for advanced, self-configuring software systems. Groups may be used for modeling various concepts, like locations, access rights, ownership, and arbitrary other classifications of devices.

Note that some of the given abstract, business-oriented concepts could be mapped to existing object-oriented and componentware concepts like class, component, or interface of current technical approaches like Java [AGH00], CORBA [Pop98], or COM+ [BBC00]. However, these approaches do not support all necessary features, especially with respect to the dynamic configuration and the description of the involved entities.

## 3 TECHNICAL REQUIREMENTS AND SOFTWARE ARCHITECTURE

Similar to the previous section, in the following we first identify the essential technical requirements for systems within the cooperative building domain. Based on this, we develop a suitable technical software architecture.

### *3.1 Technical Requirements*

The following technical requirements influence the overall solution:

*Dynamic Reconfiguration:* The system must be prepared for arbitrary adding and removing of devices or services, the entry of initially unknown devices or services, and the mobility of devices or services within or beyond the installation space.

*Robustness:* Due to the sensitive area of deployment in private homes, the requirements with regard to system robustness are very high. Potential security hazards or malfunctioning of devices caused by connecting them to the system must be avoided in order to gain and sustain user acceptance. Both the system and the design of individual devices should guarantee continued operation on a basic level, even if they are disconnected or interrupted during communication.

*Communication Bandwidth and Latency:* Typical use cases for controlling and monitoring devices and buildings may be realized with a modest communication bandwidth of 1 to 100 kbit/s. More advanced scenarios involving the transmission of audio and video data will, however, require much higher bandwidths in the area of 1 to 100 Mbit/s.

*Scalability:* Scalability is concerned with the property of the system to handle a varying number of participating devices and services. The communication infrastructure should be able to handle the increased communication load properly and to support a wide variety of different communication media, especially including wireless facilities.

*Interoperability and Standards:* It is expected that various devices from different vendors will be participating in a single installation. With regard to user acceptance and robustness, incompatibilities between these devices are not tolerable. Therefore, the system must specify and employ clearly defined interfaces and protocols of communication, and has to ensure downward compatibility once these interaction standards change. With regard to interoperability, it is highly important to use existing standards for communication and interaction whenever possible.

### *3.2 Existing Technologies and Approaches*

There are a number of existing technologies and approaches which apply to the previously described application domain. Based on the level of abstraction and the intended user group, it is possible to structure them into three categories:

*Basic Communication Infrastructure* comprises low-level technical approaches which may be employed to supply the means for inter-device communication within a building. We consider HomeStar [Tec00], PowerLine with X10 [Hoe96], CEBus [CEB00], LONWorks [Ech00], Smart House [Sma00], and the European Installation Bus [Eur99] as representative members of this group.

*Application Frameworks* and APIs are specifically suited to support application development for intelligent buildings. They define abstractions for important business entities like 'device 'or 'service 'on a higher level of abstraction and provide or specify a required technical infrastructure as well as a common programming model. Jini [Sun99], OSGi [OSG00], UPnP [Mic99], KNX [Kon01], and OWL [BPR00] are relevant examples for this group.

*Consumer Products* are commercially available off-the-shelf solutions for particular problems of the application domain. In contrast to frameworks and APIs, they are targeted at the consumer and thus do not directly apply to the presented approach.

Note that the given distinction is obviously subjective and not necessarily disjoint. A given consumer product, for example, may provide a particular application framework or communication infrastructure of its own, whereas a certain API could also be used in a different application domain.

## *3.3 Software Architecture*

Modern households are expected to incorporate intelligent appliances step by step. In particular, various different technical solutions, as categorized in the previous section, are liable to co-exist. In this section, we present a technical architecture which provides the flexibility and scalability required to cope with the requirements of a heterogeneous technical environment. Moreover, the proposed architecture also incorporates the business model discussed in Section 2 as well as its implied functional requirements.

Therefore, we propose a layered architecture [BMR96] with clear abstractions and interfaces between each of the three layers (cf. Figure 3). This architecture allows to exchange particular technical solutions on each level. The first layer represents the communication infrastructure which is encapsulated by the communication abstraction. This allows for consistent access of possibly different communication technologies. For example, a household might use both X10 [Hoe96] and the European Installation Bus [Eur00] to couple its appliances. Both technical solutions are addressed via the same, uniform interface.

Based on this foundation, the second layer implements the abstract business model as discussed in Section 2. To organize all entities, like agents, skills, or collaborations, a number of specific managers offer dedicated services. Consider, as an example, the description manager which keeps track of all descriptions in the system, specifying aspects of agents or their skills.

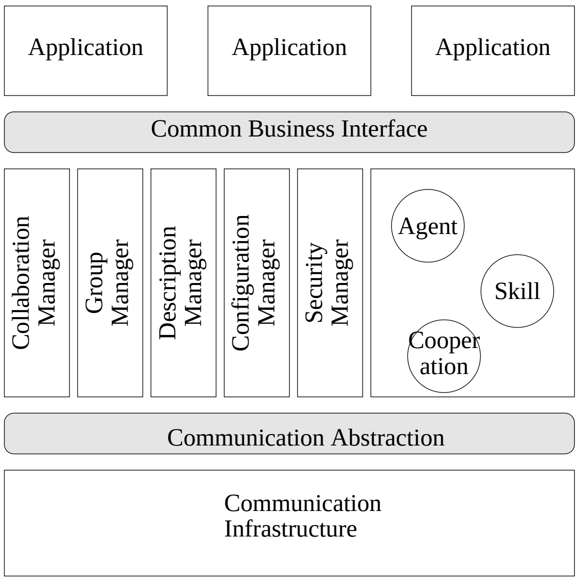

Figure 3: Technical Software Architecture

A new agent introduced to the system commits a set of descriptions to the description manager. Other agents may query these descriptions and discover the agent in question. The common business interface offers a consistent way to access the services of technical managers as well as involved business objects.

The third layer represents the various applications in an intelligent home. They model the household and its appliances by a set of specialized, cooperating agents extending the common business model on the second level while using the technical services of the proposed managers.

The technical architecture presented above fulfills important technical requirements as extensibility, scalability, and standardization (cf. Section 2.3). To facilitate understanding of the architecture, we discuss the technical managers in more detail:

### *3.1 Group Manager*

The concept 'Characterizable' captures the properties and characteristics of agents, their skills, and their collaborations (cf. Section 2). Everything that is "characterizable" may be grouped together with other characterizable entities. Groups of characterizable entities share common properties, at least the common property of being part of the same group.

*Responsibilities:* The group manager has to provide services to organize and manage hierarchical groups of characterizations. Each characterization (a description or a group) refers to a set of characterizable entities (agents, skills, and collaborations). These groups allow to combine a set of characterizable entities and describe their common properties.

As an example, consider a given instance diagram D1, containing the agents A1, A2, and A3 in a certain configuration. Agent A1 and A2 are located in the same room R1. Thus, they are both part of the group R1, which is described by a description D2. Thus, characteristics of groups are described by descriptions, but groups themselves can be part of a group again. E.g., the room R1 could be part of the building B1, which is again a group described by D3.

All in all, the basic issue of the group manager is to provide a set of hierarchies of characterizable entities. Moreover, the group manager offers services to browse and query these hierarchies. To run a query, the client has to provide a description containing the properties of the groups he is looking for. The group manager then returns all groups matching the description. As the group manager is the entry point for agents to explore the universe it is necessary to authenticate and authorize all clients. To this means, a close cooperation with the security manager (cf. Section 3.4) is mandatory.

*Implementation aspects:* Approaches like Jini [Sun99] or Universal Plug and Play [Mic99] both provide a naming service and some kind of trading service. However, a sophisticated group manager as described in this section is currently not available. New technologies like LDAP servers [HSGH98] or meta-directory services in combination with existing Naming and Trading Services [Pop98] may be used to implement the group manager. Based on the LDAP services, all characterizable entities may be able to register themselves with the directory service. Once they are registered, they may be introduced in several hierarchical groups. Additionally, meta-information may be added to each group. Thereby, the basic requirements could be realized on top of available meta-directory services. As these services are designed to be scalable, robust, flexible and highly interoperable, it seems possible to realize most of the technical requirements based on existing components in the near future.

## *3.2 Collaboration Manager*

The collaboration manager component plays a central role for the highly dynamic, cooperative nature of the system. As described in Section 2, agents use their skills to collaborate in order to perform useful tasks for the users. To this means, it is necessary to determine the relevant collaborations and establish the required connection structure at runtime.

*Responsibilities:* The collaboration manager uses the group manager to discover registered agents, their skills and defined collaborations of the system. The information returned by the group manager has to be suitable to contact the entities in question and retrieve associated descriptions from the description manager. This information is used by the collaboration manager to choose a suitable collaboration based on available skills, current state of participating agents, and applicable security policy as determined by the security manager.

A simple usage of fixed, bilateral collaborations between user agents and an individual skill is similar in result to the use of a conventional Trading Service, as specified in CORBA [Pop98]. In the energy management application example, it should be possible to enumerate all collaborations in the system that offer a simple dishwashing service. However, the use of more complex collaborations, like dishwashing in an energy-saving scenario, requires more “intelligent” behavior from the collaboration manager. It has to be able to retrieve the necessary collaboration descriptions, find suitable agents and skills, and establish the proper connection structure.

In general, this more advanced functionality of the collaboration manager requires domain-specific knowledge which may be supplied to the system in the form of dedicated descriptions. Although automatic discovery and coordination of agents or their skills is preferred, it may be necessary to ask for human assistance or allow the configuration manager to override certain choices of the collaboration manager.

*Implementation aspects:* Although the collaboration manager could be implemented as an individual, unique component of the system possibly with redundant backup components, an interesting implementation variant is a federation of different collaboration managers. In this case, each collaboration manager possesses its own, possibly partial view of the system, exchanging registration information with other known collaboration managers, and redirects queries to its partners if the individual knowledge is not sufficient. Taken to the extreme, every agent of the system implements the necessary functionality, leading to the conventional system model of agent-based artificial intelligence research. While such an approach offers an elegant solution to the problem of robustness, as demonstrated by the reliable DNS name service of the Internet, for example, the deployment, management and trouble-shooting of such a system becomes more difficult.

## *3.3 Description Manager*

The conceptual framework of Section 2 offers descriptions to allow for proper specification of dedicated aspects of agents as well as their skills, and collaborations. Everything, which assigns a characteristic to an instance of the system, might be regarded as a description. Conceivable descriptions range from lists of simple attribute-value pairs for individual skills to complete interaction specifications, e.g. message sequence charts [Kru00] for complex collaborations of several agents.
Consider, for example, a message sequence chart, which species a service protocol for energy inquiries. To offer inhabitants of an apartment statistics of their respective energy consumption, cooperating agents

are required to understand this inquiry protocol. Proper establishment of collaborations based on descriptions is the most prominent motivation for the concept of descriptions. Because all instances adhering to a description constitute a group, the description manager heavily relies upon the group manager.

*Responsibilities:* The description manager is responsible for an accurate, up-to-date grouping of all instances known in the system according to the set of descriptions. Especially, it reacts to new descriptions or characterizable entities which are introduced during run-time of the system. Thus, an existing characterizable entitiy is added to a description's group. Based on the collected information, the description manager may be queried at any time for descriptions or characterizable entities they relate to. Possible queries include, for instance, "Which descriptions match agent X?" or "Which agents fulfill descriptions Y and Z?".

*Implementation aspects:* The core functionality of the description manager is the decision whether a characterizable thing fulfills a given description. For this purpose, two realizations are conceivable and most likely used together for maximum benefit: The solution based on groups involves a simple implementation, if a sophisticated group manager is available. However, managing the possibly numerous groups in a highly dynamic system might be expensive. As the description manager solely accesses these groups, a separate implementation of the corresponding mechanisms might be more promising.

Another variant is to provide the description manager with mechanisms which allow to decide dynamically whether a description fits or not. For example, one might decide not to keep track of all door devices in an apartment but instead match a device with the description "door" if it offers the protocols "open door" and "close door".

### *3.4 Security Manager*

The security manager cooperates with all other managers and components in order to check whether their intended operations are admissible. To this means, it has to identify and authorize the participating agents and system components as well as enforce restrictions on their behavior based on certain rules. Note that we suppose the presence of sufficient base facilities in the communication infrastructure, for example, with respect to symmetric as well as asymmetric data encryption and digital signing of messages.

*Responsibilities:* As already mentioned, the primary responsibility of the security manager is to ensure that only admissible operations are performed. Furthermore, the security manager should also be able to recognize malevolent components and to take active countermeasures against them.

To this means, the security manager should block the entry to the system for unauthorized or unknown agents, components, and devices. A variety of strategies are conceivable: Manufacturer-based or individually created digital certificates may grant certain rights to specified agents, the agent itself could allow for inspection of its code by the security manager, employing techniques, like proof-carrying code [Lee00. Access may also be granted by resorting to the agents' descriptions, especially when they have been created by a trustworthy party. An example for this is a location-based security service, which allows only devices physically located within the user's apartment to participate in certain collaborations.

The security manager also supervises the ongoing operation of the system to protect the user from unwanted actions of malevolent or faulty agents. To this means, the security manager uses the description manager to achieve a characterization of the desired behavior within a collaboration and checks whether the intended actions fit with this behavior. This may be done by explicit means, like, for example, access control lists for the respective agents, or by resorting to high-level, graphical or declarative behavior specifications.

*Implementation aspects:* It doesn't seem reasonable to realize the security manager as a single, unique component, as it is involved in each and every operation of the system. A failure of this central component may eventually render the whole system useless. Furthermore, if an aggressor compromised a single, central security manager, the whole system would become vulnerable.

Therefore, a highly distributed implementation seems to be adequate, where each agent is responsible for authorizing other components and their operations. As a starting point in the design of such a system, existing security schemes based on peer structures may be analyzed, like the web of trust imposed by the public key infrastructure of the PGP program [Zim95].

### *3.5 Configuration Manager*

The configuration manager provides the interface to configure entities such as agents, skills, and collaborations, which offer a certain service to the user. Each entity provides an interface to its features so that the configuration manager may present and manipulate them in a uniform way.

*Responsibilities:* The configuration of the entities in question is based on their respective characteristics. It uses the group manager, the description manager, and the collaboration manager as helper services. Configuration of entities is needed during their whole life cycle: after the installation of a new device, for a change of behavior, such as a modifying the energy consumption policy, and after removal of a device. For each of these stages, the configuration manager has to be able to locate a new device, to contact it, to query the characteristics, and to present them with a uniform interface.

*Implementation aspects:* The configuration of the system and its constituents should be possible for any type of device from anywhere in the network, especially via direct manipulation of the device, e.g. by pressing buttons on its front panel. Therefore, every device should provide its own configuration interface that may be integrated into a container. Moreover, it is mandatory that the presented configuration interface is adaptable to different output devices like touch screens or handheld computers. This could be implemented by separating the description of the configuration interface from its presentation on a particular output device. A similar approach is used in XML [XML01], for example, which separates structure, content, and presentation of a document.

## 4 CONCLUSION

In this paper, we elaborated an software architecture for cooperative buildings which consists of two main parts: an abstract business-oriented model and a technical software architecture. Thereby, the presented solution abstracts away from technical details and concentrates on the common business model as a foundation of applications in the scope of intelligent buildings. The proposed software architecture is structured into several layers separated from each other by well-defined interfaces. This supports both flexibility and scalability of the approach.

The core business framework builds around the essential concepts *agent*, *skill*, and *collaboration*. We demonstrated how the business model may be easily mapped to existing technologies and thus may be considered as an appropriate foundation for future implementations. A number of dedicated managers is responsible for various different aspects as, for instance, the administration of groups and descriptions, or the overall system security. We discussed each of these managers in detail, reasoned about implementation variants as well as their mutual interactions. Moreover, we outlined how existing technical solutions may be mapped to the proposed architecture.

Based on this work, we recommend further interdisciplinary elaboration of the individual aspects. First of all, more scenarios of different application areas should be analyzed. We considered energy management as a prominent example, while security and health care represent constitute other highly interesting market segments. On the technical side, the evaluation of existing technologies and solutions should be elaborated to identify promising candidates for actual implementation. Building upon gained practical experience, the proposed architecture may be further revised and refined.

## REFERENCES

[AGH00] Ken Arnold, James Gosling, David Holmes. The Java Programming Language - Third Edition. Addison-Wesley, 2000.

[BBC00] Ray Brown, Wade Baron, William D. Chadwick. Designing Solutions with COM+ Technologies, Microsoft Press, 2000.

[BPR00] B. Bruegge, R. Pfleghar, T. Reicher Internet Framework for Cooperative Buildings. EIB Event 2000 Munich, Germany, 2000.

[BMR96] Frank Buschmann, Regine Meunier, Hans Rhonert, Peter Sommerlad, Michael Stal. Pattern Oriented Software Architecture, John Wiley & Sons, 1996.

[BRSV98] Klaus Bergner, Andreas Rausch, Marc Sihling, and Alexander Vilbig. A componentware development methodology based on process patterns. In PLOP`98 Proceedings of the 5th Annual Conference on the Pattern Languages of Programs. Robert Allerton Park and Conference Center, 1998.

[CEB00] CEBus Industry Council Incorportion. CEBus Industry Council Homepage. http://www.cebus.org, 2000.

[Eur00] European Installation Bus Association. EIB Handbook Series, Edition 3.0, 2000.

[Hoe96] Dan Hoehnen, X10 FAQ. ftp://ftp.scruz.net/users/cichlid/public/x10faq, 1996.

[HSGH98] Tim Howes, Mark C. Smith, Gordon S. Good, and Timothy A. Howes. Understanding and Deploying LDAP Directory Services (MacMillan Network Architecture and Development Series). MTP Press, 1998.

[Joh99] Johnson Controls. Metasys Facility Management System. http://www.johnsoncontrols.com/Metasys, 1999.

[Kon01] Konnex Association Homepage. http://www.konnex-knx.com, 2001.

[Kru99] Ingolf Krüger. Distributed System Design with Message Sequence Charts, Dissertation at the Technische Universität München, 2001.

[Lee00] Peter Lee. Proof-Carrying Code. http://foxnet.cs.cmu.edu/people/petel/papers/pcc/pcc.html, 2000.

[Mic99] Microsoft Corporation.UPnP -Universal Plug and Play Forum. http://www.upnp.org, 1999.

[OSG00] OSGi. Open Services Gateway Initiative. http://www.osgi.org, 2000.

[Pop98] Alen Pope. The Corba Reference Guide: Understanding the Common Object Request Broker Architecture. Addison-Wesley, 1998.

[Sun99] Sun Microsystems. Jini Architectural Overview, Technical White Paper, 1999.

[Tec00] Lucent Technologies.Lucent HomeStar. http://www.lucent.com/networks/homestar, 2000.

[XML01] XML Consortium. XML Homepage. http://www.xml.org, 2001

[Zim95] Philip Zimmermann. The Official PGP
User's Guide. MIT Press, 1995.