TECHNICAL UNIVERSITY OF CLUJ-NAPOCA
AUTOMATION AND COMPUTER SCIENCE FACULTY
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

SMART HOUSE – CONTEXT SENSITIVE MULTIAGENT SYSTEM

HOSZU AMALIA

2008

UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
SECȚIA CALCULATOARE

VIZAT DECAN	**VIZAT ŞEF CATEDRĂ**
Prof. Dr. Ing. Sergiu NEDEVSCHI	**Prof. Dr. Ing. Rodica POTOLEA**

SISTEM DE AGENȚI SENZITIV LA CONTEXT PENTRU O CASĂ INTELIGENTĂ

Absolvent: **Hoszu Amalia**

1. CONȚINUTUL PROIECTULUI:
   A. Piese scrise
   Introducere
   Studiu bibliografic
   Fundamentare teoretică
   Specificații şi arhitectură sistem
   Proiectare de detaliu
   Cerințe prototip
   Rezultate
   Concluzii
   Dezvoltări ulterioare
   Bibliografie
   B. Anexe
   Listingul codului sursă
   CD conținând programul şi uneltele auxiliare utilizate
2. LOCUL DOCUMENTAȚIEI: Universitatea Tehnică Cluj-Napoca
3. CONSULTANȚI: Asist. Ing. Lucia Văcariu
4. DATA EMITERII TEMEI: 01.08.2007
5. TERMEN DE PREDARE: 20.06.2008

CONDUCĂTOR DE PROIECT	**SEMNĂTURA ABSOLVENT**
Asist. Ing. Lucia Văcariu	

Table of content

List of figures

I. INTRODUCTION

This chapter presents the motivation for this thesis, research issues, challenges, and the approach I have chosen to address these issues.

The current research in the area of the context-aware computing brings forward several solutions to the problem of context adaptation of devices, services and systems to their physical and computational environments. This thesis attempts to analyze the existing research directions and the state-of-the-art technology for modeling context-aware systems. I try to identify some of the unsolved issues in this domain and to propose a flexible and scalable solution for designing and implementing a context-sensitive Smart House system.

Distributed, context aware, pervasive systems face an extensive range of challenges starting from the specific issues faced by distributed systems such as integrating and ensuring interoperation of a large number of heterogeneous devices, flexibility in integrating devices with limited resources and performance, fault tolerance, failure management and security. Context aware computing brings new challenges with respect to the complexity of capturing context from noisy and often ambiguous sensor data, and also with respect to the process of processing, semantically modeling, and disseminating contextual data.

Moreover, I argue that a new dimension of the research in this field should be centered on discovering the user intent and acting towards providing the optimum response to satisfy user’s preferences and needs. I identified several limitations in the study of the present context-sensitive systems. A first observation is related to the lack of generalization and formal basis in the context representation. These solutions are based on ad-hoc or proprietary domain representations (e.g. contextual graphs, XML, contextors). Other applications make a limited use of the ontologies in modeling the domain. Ontologies are mostly used for providing a common understanding of the modeled domain or as a common language for the entities (e.g. agents, services,) which collaborate to achieve the common goal of context adaptation. In this context, these applications do not take advantage of the expressive power provided by the ontologies, nor of the reasoning mechanism at ontology level.

Another inherent problem present in the applications that work in real domains is the uncertainty problem. Uncertainty is a common difficulty in pervasive context-aware applications, being determined in the first place by the imperfectness and incompleteness of sensor data. This results in an incorrect interpretation of low-level context information, and implicitly in inaccurate derived high-level concepts. That means that the system will work with an imprecise interpretation of the context, fact that causes an inadequate response of the system. Another aspect providing uncertainty in the context model is the inability to create a complete and exhaustive rule-based mechanism of the current context-aware systems. Additionally, reasoning with incomplete information about a concept or entity is not supported by the ontologies.

Currently, probabilistic reasoning methods based on probabilistic models have been proposed to support reasoning about the uncertainty. Efforts have been made to develop Bayesian Networks which support uncertain ontology representation, reasoning and methodologies to automatically convert from OWL ontologies to Bayesian Networks [27]. This approach takes advantage of the expressive power and efficient probabilistic reasoning capabilities of Bayesian Networks, nevertheless its shortcomings are related to the method of encoding the probability values in OWL, generating a correct representation of the Bayesian Network, and constructing the conditional probability tables.

Another important aspect is the lack of context personalization, which means the system is based on rules which govern its response to the context changes but does not have the ability to dynamically reconfigure those rules. Once the system detects the current context, it must decide what kind of behavior is desired by the user, in the given current context. However, even if the system embeds AI techniques for learning from past situations to obtain the same behavior, I have to take into consideration the situation in which *new behavior* is expected in new *or* previously learned situations. This thesis attempts to show that for building context-aware, adaptive systems, the domain knowledge and pre-defined rules are not enough in order to provide an online intelligent adaptation to the situation change and the optimum response to the user's needs or preferences.

The solution I propose is to design a distributed multi-agent system, which uses the ontology as a context representation method to its full capabilities and an innovative method for context personalization based on users' affective feedback. The essential target is to design a system which adjusts its behavior by taking into account users' satisfaction level, measured as affective response, to the systems decisions.

In the following I will present the scope and requirements of this thesis:

1. Study of existing context-representation models as well as the use of ontologies in context-aware research.

This thesis attempts to provide a comparative analysis of the design of context ontologies as well as the approaches to attach probabilistic extensions to ontologies to support uncertainty.

2. Study and investigation of several agent prototyping environments

Due to the continuous improvements and maturing of agent prototyping environments, which provide visual tools and interfaces to aid the design and realization of diverse agent based systems, an evaluation of these systems needs to be approached is a prerequisite.

3. Study of the available software agents development methodologies

Software development methodologies provide guidelines for the system designer when developing a system. As agents become widely used as a leading technology for creating complex systems, a growing need for practical methods for developing agent applications arises. From this point of view a comprehensive analysis of the current tools and methodologies for building multi agent systems has to be performed.

4. Designing the Smart House multi-agent system architecture

Decisions have to be taken regarding the type of agents that I will develop (reactive, procreative, hybrid, BDI) and the design of multi-agent system. As part of the system design phase, I have to envisage the collaborating agents, what knowledge they share, which are their particular goals, tasks or dependencies.

5. Implementing a demo application to assess the feasibility of the design.

Constructing multi-agent systems is a difficult and complex process as they integrate the challenges existing in distributed, concurrent systems, and also the additional difficulties resulted from flexibility requirements and intensive interactions. Facing these stringent requirements, I attempt to simulate the house environment in a demo application.

A full application, integrating all the requirements of a Smart House is out of the scope of this thesis due to the complexity of the research and the necessity of expertise from different domains (Expert system developers, sensor network implementers, psychology experts, Human Computer Interaction specialists).

5. A basis for future work

The multi-agent architecture and the domain analysis presented in this thesis provide starting points and motivation to utilize and extend the work in this domain.

II. STATE OF THE ART

In the last two decades, intensive research has been done in the area of intelligent buildings. Intelligent buildings increase the benefits to their occupants by means of integrated sensor systems, computer automation, information and communication systems, smart home appliance devices, and new materials.

Integrated sensor systems judge indoor and outdoor conditions of a building and its devices in order to operate as an integrated system for maximum performance and comfort. Modern buildings become a place of multilateral interaction between the inhabitants and the building entities.

The complexity of the building environment is characterized by the people in the house, depending on their activities and needs. Different needs can be observed: privacy and comfort, safety and security, quiet, light, atmosphere, as well as ubiquitous communication and entertainment. Many innovative products are entering the market for the intelligent home. The consumers are allowed to use a wide variety of online services directly from their homes.

The research in the area of intelligent buildings and context-sensitivity is already affecting the standards of our society regarding maximizing the comfort levels in our homes. There are nearly no limits to the opportunities for optimizing the convenience of the inhabitants. Many ideas have been already realized by Bill Gates' famous house: home devices such as the washing machine that can be shut down by the mobile phone; an intelligent navigator hooked up to the TV automatically downloads the favorite show; installed wireless technology enables portable devices (such as pocked-sized electronic checkbooks, book-sized reading tablets, and digital kitchen assistants) to interact [11].

2.1. Context

2.1.1. Definitions

a) Context

Context has been considered in different fields of computer science, including natural language processing, machine learning, computer vision, decision support, information retrieval, pervasive computing and more recently computer security. By analogy to human reasoning, the goal behind considering context is to add adaptability and effective decision making.

As mentioned in [7], context refers to “what surrounds the center of interest, provides additional sources of information and increases understanding”.

In a general sense, context can be defined as relevant, interrelated conditions, such as environments or situations, in which something exists or occurs.

According to Dey & Abowd in [2], context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves. Context-aware applications look at the **whos**, **wheres**, **whens**, and **whats** of entities and use this information to determine **why** the situation is occurring.

b) Context awareness

Context awareness is a specific topic that deals with how devices can infer and understand the current context based on the input given by the sensors [1]. By understanding the current context (situation, environment) the user or the device is in, the system can intelligently make a decision to do a certain action, or at the very least notify and ask the user to confirm the action to be taken.

Context-Aware Recommender Systems are a core component in these environments. The goal of these intelligent agents is to proactively suggest/prioritize items that a user may be interested in, by taking into account the context of interaction (e.g., location, cognitive load, goals).

c) Ubiquitous computing

Mark Weiser, chief scientist of Xerox PARC was the first to introduce the term ubiquitous computing and defined its characteristics. Ubiquitous computing is commonly known by its short form, ubicomp [1] and is also known as **pervasive computing**. Ubiquitous computing has these following important aspects:

- the computing devices are scattered everywhere around us;
- they are interconnected to and interact with each other;
- they do not require continuous active attention from the user in order to work;
- each device tends to be more special purpose rather than general purpose;
- they become invisible by being seamlessly integrated into the background.

When ubiquitous computing is applied to the home context, it would turn a regular home into a smart home. What the residents will notice is how the home and its smart applications are adjusting itself and its environment to constantly adapt to the residents’ dynamic needs, while requiring none to only minimal interaction between the residents and the smart home applications.

Due to its nature, pervasive computing is closely related to context. The principal reason of this correlation comes from the high heterogeneity and ubiquity of communicating entities in these kinds of environments. These two aspects require run-time adaptation of provided services and of users’ devices depending on their location, role and task at hand, where, adaptation is mainly dependent on the situation of use or ”context”.

d) Contextual information

Contextual information was classified into several categories:

- **user context** (user’s profile, user’s location, social situation, etc.);
- **computing context** (network connectivity, communication costs, communication bandwidth);
- **physical context** (lighting, noise levels).

Other authors suggested other types of context: **location**, **identity**, **time** and **activity**, out of which other conceptual information can be inferred [7].

e) The context attribute designates the information defining one element of context, e.g. “ActivityLocation”, “NamePerson”, “ActivityDuration”.

Each context attribute has at least one value at a given moment, the value depending on several entities to which the attribute relates. An entity is an instance of a “person, object or place”, but can also be an activity, an organizational concept (role, group, another agent, etc).

A single context atom can be described with a couple of attributes [10], such as:

- **Context type**: The context type refers to the category of context such as temperature, time, speed, etc. This information type may be used as a parameter for a context query or a subscription.
- **Context value**: Context value means the raw data gathered by a sensor. The unit depends on the context type and the applied sensor. In most cases, context type and context values are not enough information to build a working context-aware system.
- **Time stamp**: This attribute contains a date/time-value describing when the context was sensed. It is needed e.g., to create a context history and deal with sensing conflicts.
- **Source**: A field containing information how the information was gathered. In case of a hardware sensor it might hold the ID of the sensor and allow an application to prefer data from this sensor.

- **Confidence:** The confidence attribute describes the uncertainty of this context type. Not every data source delivers accurate information, e.g., location data suffers inaccuracy depending on the tracking tool used.

f) Contextual modeling

Contextual modeling refers to the way we can represent contextual information in a structured and meaningful manner so that it can be accessed and used by applications that perform further computations.

2.1.2. Context processing

Meaningful context information has to be derived from raw data acquired by sensors. This context processing aims at building concepts (knowledge) from environmental information sensed by sensors.

Information can be also provided directly by users, such information is called profiled information.

This type of processing is also known as context interpretation. It should contain two separated sub-steps:

- Modeling: Raw data is modeled to reflect physical entities, which could be manipulated and interpreted.
- Evaluation: Propositions from the modeling module are relative to a particular situation. They need to be evaluated against a particular context.

As soon as the raw context data is sensed by a data source, it has to be processed as the modules that use it are rather interested in already interpreted and aggregated information than in raw data.

Whereas context aggregation refers to the composition of context atoms either to collect all context data concerning a specific entity or to build higher-level context objects, context interpretation refers to the transformation of context data including special knowledge.

Several approaches related to context processing have been implemented:

1) **The Context Toolkit** [22] offers facilities for both context aggregation and context interpretation. The context aggregators (context servers) are responsible for composing context of particular entities by subscribing to relevant widgets, context interpreters provide the possibility of transforming the context.

2) In **SOCAM** [13], the Context Reasoning Engine reasons over the knowledge base, its tasks include inferring deduced contexts, resolving context conflicts and maintaining the consistency of the context knowledge base. Different inference rules used by the reasoning

engine can be specified. The interpreter is implemented by using Jena2, a semantic web toolkit.

3) In **CoBrA** the Inference Engine processes context data. The engine contains the Context Reasoning Module responsible for aggregating context information. It reasons over the Context Knowledge Base and deduces additional knowledge from information acquired from external sources [10].

4) In **CASS** deriving of high-level context is also based on an inference engine and a knowledge base. The knowledge base contains rules queried by the inference engine to find goals using the so-called forward chaining technique [10].

2.1.3. Context representation

An efficient model for handling, sharing and storing context data is essential for a working context-aware system.

Several representations of context exist: contextual graphs, XML, contextors, object-oriented models and more recently, ontology-based representations.

A contextor is a software abstraction, which models a relation between variables that represent the context, and returns the value of variables that belong to that context [24].

All these representations have strengths and weaknesses. The lack of generality is the most frequent weakness: some representations are suited for a type of application and express a particular vision on context. There is also a lack of formal bases necessary to capture context in a consistent manner and to support reasoning on its different properties.

The **Context Toolkit** [22] handles context in simple attribute-value-tuples, which are encoded using XML for transmission.

Hydrogen uses an object-oriented context model approach with a superclass called ContextObject which offers abstract methods to convert data streams from XML representations to context objects and vice versa [10].

More advanced ways of dealing with context data based on ontologies are found in SOCAM, CoBrA and the Context Managing Framework.

The **SOCAM** authors divide a pervasive computing domain into several sub-domains, e.g. home domain, office domain etc., and define individual low-level ontologies in each sub-domain to reduce the complexity of context processing. Each of these ontologies implemented in OWL provides a special vocabulary used for representing and sharing context knowledge.

CoBrA also uses an own OWL-based ontology approach, namely COBRA-Ont.

Ontology represents a formal specification of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the objects within that domain. Ontologies provide a uniform way for specifying the model's concepts, subconcepts, relations, properties and facts, also providing the means for the sharing of contextual knowledge and information reuse. The ontology reasoner evaluates the context. This allows the system to determine the contextual compatibility, to compare contextual facts and to infer new and more complex context from sensor measurements, also counteracting the common problem of incompleteness and ambiguity.

An ontology-based representation seems the best approach, because it fulfills some very important goals related to context information management.

- A first goal is defining intra-context inferences in a clear and unambiguous way, helping in interpreting data coming from sensors or observations;
- Another goal is to exploit reasoning on context ontologies to overcome partial and imprecise context information. This may be useful when the detection of contextual data is imprecise or incomplete (e.g., detecting whether a user is moving or not, his/her mood, or whether he/she is with other people may be difficult);
- A further goal is the possibility to share and exchange context information among different recommenders in order to integrate these data and make more precise and complete the information about the user current context conditions.

Various context-aware frameworks use ontologies as underlying context models. The conclusions of the evaluation presented in Strang and Linnhoff-Popien [21], based on six requirements, show that ontologies are the most expressive models and fulfill most of their requirements.

2.1.4. Context adaptation

The process of making use of contextual information in order to change the behavior or characteristics of a context-aware application is commonly known as *context adaptation*. In general there are two ways a context-aware application can adapt to the new or changing context.

- Self action or self reconfiguration:

When an application becomes aware of a new context it can adjust its *own* characteristics relative to its user's predefined or learned preferences for that context. In some cases, the application developer can make some assumptions about the user's preferences, or the application itself can employ some machine learning techniques to learn about its user's preferences. The preferences could be modeled as profiles [12] to make it possible for the user to group together related preferences in a situation, and for the user and the application to distinguish between sets of preferences in different contexts.

- Other action:

Instead of selecting profiles or adjusting the device's own settings based on the user's preferences, a context-aware application could also trigger other actions to be performed which do not involve changing anything on the device itself. Examples of such actions include sending notifications (e.g. via SMS, email), grabbing user's attention by beeping, playing a tone, or displaying a warning message, sending messages to other applications.

2.1.5. Learning and reasoning

Several approaches have been proposed recently concerning multi-agent learning in context aware systems. Learning how to choose among context attributes which are the relevant ones could be very important in an application where the amount of available context information is too large and the effort needed to compute the values for all those attributes is not acceptable in term of efficiency. The solution proposed in [6] was to combine individual learning method (in connection to user's feedback) and also by a collective one. For individual learning (mono-agent learning) on how to choose among context attributes those who are relevant for a given situation, the agent uses the feedback from the user.

The knowledge sharing between agents improves the method of individual learning so that if an agent does not know which attributes could be relevant for the situation in which it is, probably for the first time, it can ask other agents which are the attributes which they already know as being relevant. In the same way, if an agent had not had a lot of feedback from its user on attributes in a specific situation, it can again try to improve its set of relevant attributes, by asking other agents.

2.1.6. Context sharing

In a ubiquitous computing environment there will be potentially many contextual information acquired by many heterogeneous devices. The contextual information will in turn be managed by many different special purpose applications that coexist within the ubiquitous computing environment like a context-aware smart home. Some of these different devices and/or systems need to talk to each other for sharing information or to notify other applications regarding a certain condition that other applications may be interested in. Therefore it is important for these applications to agree on a common standard for modeling contextual information and probably other relevant non-contextual information that needs to be shared. This means there are at least two parts to the standard:

- for **modeling** the contextual information;
- for **communication protocol**, which could be between the applications using the same or different standards for modeling contextual information.

Many applications may be interested in being notified about changes of context. Typically, the context source monitor polls the current context and sends the changes to some context service that has a publish-subscribe notify interface. The context service is responsible to deliver the context changes to the clients who have subscribed to the related context changes. Different context, however, has different properties. This difference leads to the necessity of a different polling rate on different context.

2.1.7. Security and privacy

As context may include sensitive information on people, e.g., their location and their activity, it is necessary to have the opportunity to protect privacy.

For these purposes the Context Toolkit [22] introduces the concept of context ownership. Users are assigned to sensed context data as their respective owners. They are allowed to control the other users' access. The Owner Permission is the component that receives permission queries and determines to grant or to deny access based on stored situations. These situations include authorized users, time of access, etc. Now applications and components have to provide their identity along with the usual request for information. Finally the authenticator is responsible for proofing the identity by using a public-key infrastructure.

CoBrA includes an own flexible policy language to control context access, called Rei [10]. This policy language is based on the concepts of rights, prohibitions, obligations and dispensations and controls data access through dynamically modifiable domain dependent policy rules.

2.1.8. Keeping the history of contextual information

Chen and Kotz noted the importance of keeping a history of the acquired contextual information [14]. Depending on the application and the context, it is probably ideal that only a subset of the collected data is stored due to storage space limitations. For example, when an application is only interested in a context *change*, then the application need only to store the last data collected to be compared with the current condition (certain applications need to take the average over a certain number of samples) in order to determine that there is a change in the context. However, in other applications, for example the ones that require learning/training, it may be necessary to store many samples of various contextual information.

Such context histories may be used to establish trends and predict future context values. The Context Toolkit, CoBrA, CASS, SOCAM and CORTEX save sensed context data persistently in a database. A further advantage of using a database is the use of the Structured Query Language (SQL) which enables to read and to manipulate operations at a high abstraction level. In the CoBrA and the CASS architecture the persistent storage is called Context Knowledge Base. Additionally a set of APIs is offered to assert, delete, modify and query the stored knowledge. CASS uses its database not only to save context data but also to store domain knowledge and inference rules needed for creating high-level context.

2.2. Architectures of context-aware applications

Context-aware systems can be implemented in many ways. The approach depends on special requirements and conditions such as the location of sensors (local or remote), the amount of possible users (one user or many), the available resources of the used devices (high-end-PCs or small mobile devices) or the facility of a further extension of the system.

Furthermore, the method of context-data acquisition is very important when designing context-aware systems because it predefines the architectural style of the system at least to some extent.

Chen [15] presents three different approaches on how to acquire contextual information.

- *Direct sensor access*

This approach is often used in devices with sensors locally built in. The client software gathers the desired information directly from these sensors, i.e., there is no additional layer for gaining and processing sensor data. Drivers for the sensors are hardwired into the application, so this tightly coupled method is usable only in rare cases. Therefore, it is not suited for distributed systems due to its direct access nature, which lacks a component capable of managing multiple concurrent sensor accesses.

- *Middleware infrastructure*

Modern software design uses methods of encapsulation to separate e.g., business logic and graphical user interfaces. The middleware-based approach introduces a layered architecture to context-aware systems with the intention of hiding low-level sensing details. Compared to direct sensor access this technique eases extensibility since the client code has not be modified anymore and it simplifies the reusability of hardware dependent sensing code due to the strict encapsulation.

- *Context server*

The next logical step is to permit multiple clients access to remote data sources. This distributed approach extends the middleware-based architecture by introducing an access managing remote component. Gathering sensor data is moved to this so-called context server to facilitate concurrent multiple access. Besides the reuse of sensors, the usage of a context server has the advantage of relieving clients of resource intensive operations. As probably the majority of end devices used in context-aware systems are mobile gadgets with limitations in computation power, disk space, etc., this is an important aspect. In return one has to consider about appropriate protocols, network performance, quality of service parameters etc., when designing a context-aware system based on client-server architecture.

As pointed out in [2], a separation of detecting and using context is necessary to improve extensibility and reusability of systems.

A layered conceptual framework for context aware systems is presented in fig. 2.1.

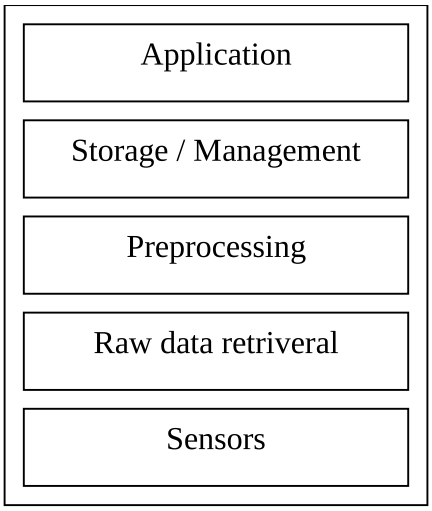


Fig. 2.1. Layered conceptual framework for context-aware systems

A **Preprocessing layer** is not implemented in every context-aware system but may offer useful information if the raw data are too coarse grained. The preprocessing layer is responsible for reasoning and interpreting contextual information. The sensors queried in the underlying layer most often return technical data that are not appropriate to use by application designers. Hence this layer raises the results of layer two to a higher abstraction level. The transformations include extraction and quantization operations. For example, the exact GPS position of a person might not be of value for an application but the name of the room the person may be in.

In context-aware systems consisting of several different contexts data sources, the single context atoms can be combined to high-level information in this layer. This process is also called “aggregation” or “composition”. A single sensor value is often not important to an application, whereas combined information might be more precious and accurate. In such a way, a system is able to determine, e.g., whether a client is situated indoor or outdoor by analyzing various physical data like temperature and light or whether a person is currently attending a meeting by capturing noise level and location. To make this analysis work correctly a multitude of statistical methods are involved and often some kind of training phase is required. Obviously, this abstraction functionality could also be implemented directly by the application. But due to a couple of reasons this task should better be encapsulated and moved to the context server. The encapsulation advances the reusability and, hence, eases the development of client applications. And by making such aggregators remotely accessible the network performance increases (as clients have to send only one request to gain high-level data instead of connecting to various sensors) and limited client resources are saved.

The problem of sensing conflicts that might occur when using several data sources has to be solved in this layer as well. Often this conflict is approached by using additional data, like time stamps and resolution information.

A **Storage and Management** organizes the gathered data and offers them via a public interface to the client. Clients may gain access in two different ways, synchronous and asynchronous. In the synchronous manner the client is polling the server for changes via remote method calls. Therefore, it sends a message requesting some kind of offered data and pauses until it receives the server’s answer. The asynchronous mode works via subscriptions. Each client subscribes to specific events it is interested in. On occurrence of one of these events, the client is either simply notified or a client’s method is directly involved using a call back. In the majority of cases the asynchronous approach is more suitable due to rapid changes in the underlying context. The polling technique is more resource intensive as context data has to be requested quite often and the application has to prove for changes itself, using some kind of context history.

The client is realized in the **Application layer**. The actual reaction on different events and context-instances is implemented here. Sometimes information retrieval and application specific context management and reasoning is encapsulated in form of agents, which communicate with the context server and act as an additional layer between the preprocessing and the application layer [15].

2.3. Frameworks to develop context-aware applications

The following frameworks are general-purpose frameworks that can be used as a foundation to develop context-aware applications.

a CAPpella [16] is a programming by demonstration context aware framework geared towards end users instead of programmers. This framework uses machine learning techniques to recognize behavior or situation. End users demonstrate a behavior or situation of interest to the system that may have video cameras, microphones, RFID readers and other kinds of sensors. After the demonstration is finished, the user prunes the captured data in the system so that only relevant information is used by the system for learning. Then the user uses that captured information to train the system. The user needs to repeat the training process a number of times so that the system can get better recognizing the behavior or situation. The user can test how well the system performs after a demonstration is finished, then the user can decide whether to train the system with the recently captured data.

Java Context Awareness Framework (JCAF) [17] is a J2EE inspired Runtime environment and programming framework designed for developing context-aware applications. The event based runtime environment aims to support distributed and coordinated services, access control, and modular infrastructure to allow for extensions. Some unique features of the JCAF API are that it aims to be semantic free (generic purpose), and that it takes quality of context information into account. JCAF provides event listeners that applications can subscribe to. The event listeners will call the appropriate callback functions whenever there is an interesting event, e.g. a context change. What to do with the context information is left up to the applications. In other words, JCAF does not have AI built into it.

Mayrhofer's context prediction framework [18] focuses on recognizing and predicting high level context information from low level sensor data. The framework is intended to run locally in limited resource devices without requiring an infrastructure before hand. Some unique features of this framework are that it uses heterogeneous feature vectors for context recognition. It uses Growing Neural Gas algorithms for machine learning and Active LeZi algorithm for context prediction.

2.4. Context sensitive agents

The most recent approaches to context-aware pervasive applications have entailed the conclusion that the multi-agent system paradigm is a very strong candidate for implementing ambient intelligent systems. The intelligent agent paradigm is correlated to the concept of context-aware systems by the fact that each agent possesses incomplete knowledge in system with no global control, with decentralized data and asynchronous computation.

Multi agent technology is the most common deployed computational paradigm for creating systems that are flexible, can easily adapt to environmental changes, and are able to integrate heterogeneous components, by providing the flexibility to manage large scale, complex, distributed systems.

2.4.1. Motivations

Context aware applications are usually developed as computationally intensive environments based on distributed components. Such systems exhibit the pervading challenges that must be taken into account by the developers, from limited computation resources (especially for mobile devices), limited network connection and bandwidth, and user personalizes service. Also pervasive computing applications have to deal with integrating a large number of heterogeneous devices with limited resources and software, distributed components based on through complex heterogeneous networks, often crossing boundaries of multiple distributed systems. Mobile agent technology has become a promising tool to overcome these limitations. Agents offer adaptability reactivity, flexibility, increased system robustness and the potential for graceful system expansion.

It is widely acknowledged that mobile agent' capabilities of discovering, extracting, interpreting and validating context will significantly increase flexibility, efficiency, and feasibility of pervasive computing systems. Intelligent context-aware mobile autonomous software agents represent the response to the challenge of high complexity in ubiquitous computing domain. These systems should provide the migration facility of the agents if the computational task or the resources require this, also it should enable the agents to explore and discover context information, resources, services, and to communicate with other software agents.

2.4.2. Objectives

The ever increasing number of agent platforms varieties, has lead to the necessity of evaluating these platforms based on some key criterions [40] [41]:

- Compatibility with existing standards (agent technology are FIPA, (OMG) MASIF);
- Support for agent mobility: strong (ability of system to migrate code and execution state of executing unit), weak (migration of code only);
- The platform must run on mobile devices;
- The platform should be open source, as there is a need to integrate new functionality to the existing code;
- Programmable agent development platform;
- The agent platform supports the use of representational languages;
- It should be OS independent and support the OS variations for variations for portable devices;
- The programming language supported must be a fully portable object oriented language;
- Existent documentation, specialized forums, and reported problems.

2.4.3. The agent paradigm

Research in the area of agent systems has provided a variety of definitions to the notion of agent. However, two main common usage of the term agent can be identified: the weak notion of agency and the strong notion of agency. The weak notion of agency represents the sum of features on which most researches agree on, while the stronger notion of agency is more controversial and a subject of active research.

The strong notion of agency extends the weak notion, by adding additional properties such as belief, desire, and intention [29].

Russell and Norvig's provide a general agent definition "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors." But they specify that "the notion of an agent is meant to be a tool for analyzing systems, not an absolute characterization that divides the world into agents and non-agents." [42].

Other definitions include the differentiation of agents from regular programs or applications: The KidSim Agent stresses this differentiation: an agent is "a persistent software entity dedicated to a specific purpose". "Persistent" distinguishes agents from subroutines;

agents have their own ideas about how to accomplish tasks, their own agendas. “Special purpose” distinguishes them from entire multifunction applications; agents are typically much smaller [47].

The Hayes-Roth Agent definition enhances the notion by specifying the agent’s reasoning ability. Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions. [48].

The Wooldridge - Jennings Agent highlights some of the key agent characteristics, compatible with the weak notion of agency: “a hardware or (more usually) software-based computer system that enjoys the following properties:

- autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;
- reactivity: agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
- pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative” [49].

Other definitions make clearer the boundary between agents and common programs.

The SodaBot Agent "Software agents are programs that engage in dialogs, negotiate and coordinate transfer of information.” [51].

The Brustoloni Agent definition draws an ever more concrete boundary between agents and software programs, by focusing on the placement of the agent in a real world environment: “Autonomous agents are systems capable of autonomous, purposeful action in the real world” [51]. All these variations for the agent definitions show that there is no standard, common acknowledged vision on this issue.

2.4.4 The BDI agent paradigm

The BDI-model (Belief-Desire-Intention), formalized by the Anand Rao and Michael Georgeff's [33] is based on the three mental state concepts of Belief, Desire, and Intention. It has the philosophical basis in the Belief-Desire-Intention theory of human practical reasoning, conceived by Michael Bratman. The BDI-model naturally fits to agents that are goal directed, communicate with others and can proactively react to environmental changes.

Beliefs represent the agent’s knowledge about the current state of the environment (state of the world, the agent’s own internal state, information about other agents). In real environments, with a high degree of uncertainty, agents’ beliefs are incomplete or inaccurate. In the BDI model an important aspect is emphasizing the difference between the term belief and the term knowledge. While knowledge is assumed to be true, the agent beliefs may not necessarily be true. This shows the dynamic nature of the agent information about the world. At the agent level, beliefs provide a domain-dependent abstraction of relevant entities, thus each agent has a personal, partial view on the environment in which it operates.

Desires (goals) are the states of affairs, which the agent would like to accomplish or bring about. They should be consistent with one another.

Intentions represent the deliberative state of the agent, the activities, which the agent has committed for execution, to satisfy a desire. In a certain situation an agent may choose among a set of achievable plans. The selected plan and the commitment to execute that plan become an intention.

The BDI architecture has been implemented and successfully used in a number of complex applications, the most well-known representative being the Procedural Reasoning System (PRS). For the real, concrete systems, the abstract notions of desires and intentions are converted to the more concrete concepts of goals and plans.

The goal is a central concept of the BDI architecture, representing a certain target state that the agent is trying to reach. **Goals** are the chosen desires that an agent tries to achieve, if they believe that they are achievable in the current situation.

When certain goals fail, the agents can determine if the target state is already achieved, and can deliberate whether to retry the failed action, or try out another set of actions to achieve the target state. Moreover, the goal concept allows to model agents, which are not purely reactive, and to develop agents, which exhibit pro-active behavior.

Plans represent the means of achieving a goal, the possible sequence of actions to execute in order to reach a goal. **Intentions** are a set of plans the agent tries to execute in order to accomplish a goal.

A plan is not just a sequence of basic actions, but may also include sub-goals. Other plans are executed to achieve the sub-goals of a plan, thereby forming a hierarchy of plans. The agent keeps track of the actions and sub-goals carried out by a plan, to determine and handle plan failures.

The BDI model has proved to be highly effective to model agents that run in dynamic environments and that can operate in a flexible manner despite incomplete information about the environment and the other agents.

2.5. Applications in context-aware systems

Context Ontology Language (**CoOL**) is based on the development of ontologies for specifying the model’s core concepts as well as an arbitrary amount of subconcepts and facts, altogether enabling contextual knowledge sharing and reuse in an ubiquitous computing system [15]. This contextual knowledge is evaluated using ontology reasoners.

The **CONON** context modeling approach by Wang [19] is based on the idea of developing a context model based on ontologies because of its knowledge sharing, logic inferencing and knowledge reuse capabilities. Wang developed an upper ontology, which captures general features of basic contextual entities and a collection of domain specific ontologies and their features in each subdomain. The CANON ontologies are serialized in OWL-DL, which has a semantic equivalence to well researched description logic. This allows for consistency checking and contextual reasoning using inference engines developed for description languages.

A promising context modeling approach based on ontologies is the **CoBrA system** [20]. This system provides a set of ontological concepts to characterize entities such as persons, places or several other kinds of objects within their contexts. The CoBrA system uses broker-centric agent architecture to provide runtime support for context-aware systems, particularly in Intelligent Meeting Rooms, a prevalent scenario of a ubiquitous computing environment. The context broker consists of four functional main components: the Context Knowledge Base, the Context Inference Engine, the Context Acquisition Module and the Privacy Management Module.

Another attempt for developing a reusable solution for context acquisition is the **Context Toolkit** [22]. It is a generic Java framework that enables rapid prototyping of context-aware applications. The Context Toolkit is inspired from GUI toolkits in the way they mediate between the application and the user. Mediation in the context toolkit is performed between the application and the sensed information. It is realized by a set of abstract components (widgets, interpreters and aggregators) that support the acquisition of context data from sensors and to process it into high-level context information. A widget is a software component that provides access to context information through a uniform interface. It hides the complexity of sensors and provides reusable building blocks for context sensing.

Applications interested in specific context information may subscribe to the corresponding widgets. The interpretation problem is solved by the mean of interpreters. An aggregator is responsible about the context of a particular entity (person, place or object) and obtains information from multiple widgets.

One further extensible centralized middleware approach designed for context-aware mobile applications is a project called **CASS** (Context-Awareness Sub-Structure) [23]. The middleware contains an Interpreter, a ContextRetriever, a Rule Engine and a SensorListener. The SensorListener listens for updates from sensors, which are located on distributed computers called sensor nodes. Then the gathered information is stored in the database by the SensorListener. The ContextRetriever is responsible for retrieving stored context data. Both of these classes may use the services of an interpreter. The ChangeListener is a component with communication capabilities that allows a mobile computer to listen for notification of context change events. Sensor and LocationFinder classes also have built-in communications capabilities. Mobile clients connect to the server over wireless networks.

2.6. Challenges and open issues

One of the key difficulties in creating useful and robust ubiquitous, context-aware applications is developing the algorithms that can detect context from noisy and often ambiguous sensor data.

The challenge for such systems lies in the complexity of capturing, representing and processing contextual data. In addition to being able to obtain the context-information, applications must include some “intelligence” to process the information and to deduce the meaning. This is probably the most challenging issue, since context is often indirect or deducible by combining different pieces of context information [8].

Salber, Dey & Abowd [22] describes the following difficulties in using context information in computing applications:

- The context information is often acquired from unconventional sensor. The term unconventional refers to sensors, which are not commonly found in day to day computing environments. Examples: GPS, biosensors, pressure sensitive floors. Unconventional sensors are not easily accessible because they may not be easy to find, and may require a relatively high cost to obtain.
- Low level data acquired from the sensors needs to be abstracted into a high level representation in order to be useful. For example, temperature reading in degree

Celsius may need to be abstracted into cold, cool, medium, warm, hot. An application may need to abstract a higher level information using two or more sensor data, for example, in addition to temperature the may need humidity information to determine a room's comfort level.

- The context information may be acquired from multiple distributed and heterogeneous sources. It is uncommon that a context-aware application requires more than one sensor in order to understand a certain context. The temperature and humidity example above can be used again here.
- Context information is dynamic in nature. A context-aware application often needs to detect changes in real time and adapt to constant changes. For example, a mobile location based application being used while in motion needs to handle the constantly changing location of the user and adapt accordingly in real time by giving the user the needed information most relevant to the current location.

In addition to these difficulties, there are other challenges associated with developing context-aware applications.

- Understanding of user intent

This process is often crucial during context adaptation phase. Once a context-aware application understands the current context, it must decide how to adapt to that context best. An application can adapt itself (self reconfiguring), or adapt its own or the user's environment (send messages to other application). The application has to make its best guess on what kind of behavior is desired from the application by the user, given the current context. Some of the AI techniques used in context-aware applications relies on learning from past situations to result in the *same behavior*, however, the application may not do well when a *new behavior* is expected in new *or* previously learned situations.

- Multidisciplinary

Building context-aware systems, such as smart homes to support independent living, usually involves and requires knowledge in computer science, psychology, Human Computer Interaction (HCI), architecture, mechanical and electrical engineering, social sciences, and maybe others. Depending on the domain of the context-aware application, an expert knowledge in the domain may also be required. A team of many people with diverse backgrounds, including cultural backgrounds normally does the development of such systems.

- Cost

Some of the devices, such as biosensors, used to build context-aware computing infrastructure are not common, so they generally cost more than other common computing devices. This creates a barrier of entry to people who wants to develop context-aware applications that require specific hardware or infrastructure support. Hence the design of applications is often dictated by what devices are easily accessible to the developers.

To ensure the components interoperability, portability, flexibility and robustness of the overall developed architecture, in the process of context aware pervasive systems built on a multiagent platform basis, some problematic issues have to be analyzed and the appropriate decisions must be taken. Some of the most common problems or design decisions encountered in the development of ubiquitous systems are mentioned in [34].

A key issue in a limited resource environment is deciding how to manage the distribution the different system components between the client and server. An obvious but sensible solution would be to load most computational processes to the server, but without careful balancing this could result in quick server overloading. The best approach would be to shift the balance dynamically, by providing the powerful functionality of agent migration. A significantly increase in performance can be realized if the agents can become aware of the moment when it is appropriate and safe to migrate to client for performing some computational task. A major concern is that the agent migration is both robust and efficient.

To improve performance and overcome the latency problem in network transfers, a data pre-caching mechanism is suggested in [34].

Another issue to be taken into account, especially in poor resources mobile devices is the capability of dynamic agent tuning through task scheduling and by integrating mechanisms to support agent negotiations for efficient resources utilization.

Choosing the operating systems for mobile devices is also an important issue. The analysis performed in [34] indicated that SavaJe and Linux offer greater speed and stability while WindowsCE / PocketPC are more widely supported by hardware manufacturers.

Constructing multiagent systems is a difficult and complex process as they integrate the challenges existing in distributed, concurrent systems, and also the additional difficulties resulted from flexibility requirements and intensive interactions. The main problems regarding multiagent systems development are emphasized in [21]: the necessity of a proven methodology enabling designers to clearly structure applications as multiagent systems and the deployment of general case industrial-strength toolkits.

III. THEORETICAL FUNDAMENTALS

3.1. Towards developing a formal context model

The integration of context-awareness in pervasive systems proved to be a complex and time-consuming task due to the lack of an adequate infrastructure support [14]. Previous works in ubiquitous computing were oriented mainly towards realizing application-specific context aware systems. The main drawbacks of these “ad-hoc” system design manners are the dependency on the underlying hardware and operating systems. Also, they lack in providing a common context model, which would enable context sharing and context reasoning.

According to [13], an infrastructure for building context-aware services requires the following components:

- A common context model that can be shared by all devices and services. This component is necessary, as each interacting part of the system needs to share the same interpretation of the exchanged data, as well as a shared understanding of the data. By using a shared context model, context information from a single domain or across several domains can be efficiently changed.
- A set of services that perform context acquisition, context discovery, context interpretation and context dissemination. The SOCAM architecture proposed in [13] supports acquiring various contexts from different context providers, interpreting contexts through context reasoning, and delivering contexts in both push and pull modes.

3.1.1. The use of ontologies in context-aware research

In the early phases of research in pervasive computing systems, only a weak and limited support for knowledge sharing and reasoning was achieved. The cause of this weakness is the fact that they are not built on a foundation of common ontologies, with explicit semantic representation [25]. Latest research uncovered great potential in employing ontologies as a means of storing the knowledge related to context.

The term ontology was first introduced as a philosophical concept, referring to the subject of existence, but it has become widely used in the AI domain, representing the description of concepts in a domain. Ontologies represent complete or partial knowledge about concepts, their attributes and interrelationships [26].

An ontology-based approach is the best choice if we are trying to achieve independence with respect to programming language, underlying operating system and middleware. The advantages behind modeling the context based on the use of ontologies are the following:

- They enable formal analysis of the domain knowledge by context reasoning;
- Through reasoning high-level concepts can be derived from low-level concepts;
- The reasoner can answer queries based on the deployed ontology;
- The reasoner can check the consistency of one ontology but also can validate if an ontology is consistent with respect to other ontologies;
- Inter-ontology relations and implicit hierarchies based on rules can be asserted by the reasoner [26].

The next sub-chapters will be focused on analyzing the manner in which the ontologies were designed and integrated into dynamic pervasive environments. But the most important issue of this comparative-analysis of these systems is to evaluate the performance of the infrastructures in terms of ontology design, context reasoning and context discovery.

From the analysis of the ontology specification developed in recent projects such as CoOL [26], SOUPA [25], SOCAM [13], and others, I will try to answer to following questions:

- How well is the ontology able to represent concepts, attributes and relations in an exact and well-defined manner? [26]
- How well is the language capable to create effective queries towards the reasoner? [26]
- What is the most efficient design in the ontology architecture?
- What are the possible extensions for these ontologies to support uncertainty?
- Are the performance results in terms of context reasoning and context discovery challenging?

3.1.2. The design of context ontologies

There is a tendency towards a layered, hierarchical architecture approach for the design of context ontologies. The **SOCAM** [13] architecture is based on a two-layer hierarchy: common upper layer ontology for general concepts and the domain specific ontologies, which are applicable to several smart domains (fig. 3.1.). The upper layer ontology describes general, basic concepts for all domains, and it is shared across multiple domains. The lower layer ontology is represented by low-level ontologies, which aim to provide concepts specific for each sub-domain. The ontologies are written in OWL, the web ontology language proposed by the W3C’s Web Ontology Working Group. Ontologies defined in OWL can be referenced from web servers by other ontologies and downloaded by applications that use ontologies. The fact that the ontologies can be retrieved from different web servers, while being created by different persons provides the flexibility and extensibility required in distributed systems [26].

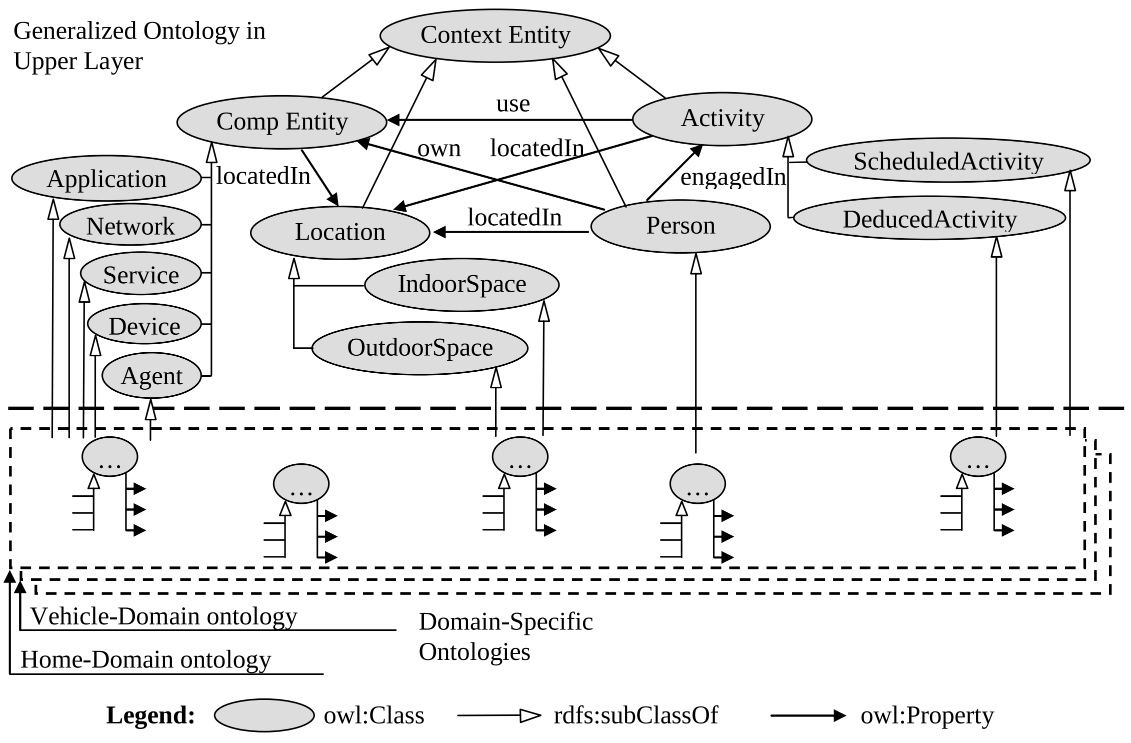


Fig. 3.1. SOCAM architecture

The innovation behind this layered architecture resides in the fact that the amount of context knowledge is reduced and the reasoning engine’s context processing overhead is limited by dynamically reloading the specific low-level ontology when the environment changes. The specific low-level ontology “re-binds” with the generalized ontology and is used to derive high-level concepts [13].

CoOL (Context Ontology Language) [26] architecture also adopts a two layers architecture: CoOL Core (based on several languages: OWL, DAML + OIL, F-Logic) and CoOL Integration, a collection of schema and protocol extensions. Authors argue that the choice of several languages for knowledge representation is favorable for developers, which may use any of the languages they consider appropriate. A reasoner, which supports F-Logic, was used, mainly because F-Logic is an expressive, highly object-oriented language, having a compact syntax and a rule based extensibility. The context model proposed is based on the ASC model (Aspect-Scale-Context). The ASC model is built around three core concepts: aspect, scale and concepts, the entities being characterized by Context Information instances. An aspect aggregates one or several scales and encapsulates objects having the same semantic type. The scale aggregates one or more context information, defining the range of valid context information. Context information encapsulates content data and the meta data which characterizes content data. By using the meta information a higher order context information characterizes the content and quality of a lower level context information. This model provides the means to define several scales for the same aspects or to derive new scales from existing ones.

SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) ontology is also expressed using OWL, the knowledge representation language for defining and instantiating ontologies [25]. As in the SOCAM project, SOUPA contains two sets of ontologies: SOUPA Core-defining general vocabularies for different ubiquitous computing applications and SOUPA Extensions - representing ontologies, which define additional concepts used in specific applications.

SOUPA Core ontologies are representations of intelligent agents with associated beliefs, desires, and intentions, time, space, events, user profiles, actions, and policies for security and privacy. The strategy used by the SOUPA developers for the implementation of such a wide range of ontologies was to adapt existing ontologies, by borrowing terms from these ontologies. These terms are not directly imported. Furthermore due to the need of interoperability between the SOUPA applications and other ontology applications, the borrowed terms in SOUPA are mapped to the foreign ontology terms using the standard OWL ontology mapping constructs. The imported terms are borrowed from well known ontologies such as: the Friend-Of-A-Friend ontology (FOAF), DAMLTime and the entry sub-ontology of time, the spatial ontologies in OpenCyc, Regional Connection Calculus (RCC), COBRA-ONT, MoGATU BDI ontology, the Rei policy ontology [25].

The SOUPA Extension ontologies represent an extension of the SOUPA Core ontologies set. SOUPA Extension provides models of ontologies, which extend the Core ontologies, thus supporting pervasive context-aware applications in smart spaces and peer-to-peer data management in pervasive computing environments.

Table 3.1. makes a comparison between SOCAM, SOUPA and CoOL.

Table 3.1. Context ontology design comparison

	SOCAM [13]	**SOUPA [25]**	**CoOL [26]**
Ontology language	OWL	OWL	OWL and DAM+OIL F-Logic
Ontology model specific issues	- it models context classification and dependency relationships associated with data types and objects - probabilistic extension, a probability value attached to each context predicate - two layer ontology design	- imports terms from known ontologies such as: The Friend-Of-A-Friend ontology (FOAF), DAML-Time ontology, The spatial ontologies in OpenCyc, Regional Connection Calculus (RCC), COBRA-ONT, MoGATU BDI ontology, The Rei policy ontology	- based on the Aspect-Scale-Context model - integrates quality aspects for context information - the ASC model can be used as a transfer model to for other context models approaches
Reasoning types	- ontology reasoning - rule-based reasoning	- ontology based reasoning	- OntoBroker used as inference engine -query language: F-Logic
Conclusions	- the overhead for loading the domain specific ontology and merging it with the upper layer ontology is low - the logic reasoning is a computationally intensive process; it may become the bottleneck when applied to pervasive computing domain. - still it is acceptable for running non-time-critical context-aware applications	- the project aims to develop a shared pervasive computing ontology - SOUPA project is a step towards the standardization of a shared ontology for pervasive computing applications	- the ASC model is flexible with respect to the application domain - ASC provides an umbrella vocabulary to transfer arbitrary context models [30] - the ASC model has the capability to deal with heterogeneous set of values describing the same entities context information

Ontologies are widely recognized as the best tool for context-modeling. The analysis [9] was based on six most relevant requirements for pervasive ubiquitous applications: distributed composition, partial validation, richness and quality of information, incompleteness and ambiguity, level of formality, and applicability. Among the other context modeling approaches: key-value, markup scheme, graphical, logic based, object-oriented, and ontology-based models, the ontological approach exhibits better modeling facilities, while providing clear model theoretic semantics.

3.1.3. Probabilistic extensions to ontologies to support uncertainty

Uncertainty is an inherent aspect in any real domain, and thus it is prevalent in pervasive context-aware applications. The most obvious cause of uncertainty is the imperfectness and incompleteness of sensor data. Moreover, context-aware applications have to support uncertain reasoning in a flexible and effective manner. Another aspect providing uncertainty in the context model is the inability to create a complete and exhaustive rule-based mechanism of the current context-aware systems. This results in erroneous interpretation of low-level context information, and implicitly in incorrect derived high-level concepts. As emphasized in [27] ontologies lack the ability to support reasoning with incomplete information about a concept or entity in the domain. Partial match of concepts within several ontologies is also a source of inaccuracy in the reasoning process.

Modeling uncertainty in context-aware systems has recently been a direction of active research. Probabilistic reasoning using a probabilistic model has been proved to offer a strong support for reasoning about the uncertainty.

A Bayesian network represents a full joint distribution over a set of random variables [29]. It has several important advantages when combined with the ontology-based approaches: can answer queries relative to its variables given the value of the evidence variables, it provides different forms of reasoning: prediction (reasoning from cause to result), abduction (inferring cause from result) and explaining away (the evidence of one cause reduces the possibility of another cause given the evidence of their common results). In a Bayesian Network the dependency relationships between the random variables is represented by the network topology itself and the uncertainty of these relationships is expressed by the conditional probability tables associated with each variable. The conditional probability tables encode the full joint probability distribution of all the variables in the domain in a compact and efficient manner.

One of the first researchers to address this issue was Ranganathan, who used Microsoft's Belief Network to create a Bayesian Network, which is mapped to the predicates in the ontology by the developer [29]. A confidence value is attached to each predicate in the ontology. Another attempt to support uncertainty in pervasive computing was to map each context predicate into a node in the Bayesian Network. If there is a dependency between two predicates, this is represented as an arc between the two nodes. These two approaches seemed to have solved the uncertainty issue, but the main drawback of their solution is that their support for uncertainty is application-specific. Their models lack reusability, so that, for a new application, the Bayesian network has to be redefined, even if the applications have similar domains. A common problem to both approaches is that there is no systematic method to support uncertain reasoning mechanism.

Binh An Troung and YoungKoo Lee, developed a "unified context model", based on the Probabilistic Relational Model developed by Friedman and on Koller's Probabilistic Frame-based systems [29]. The Unified Context Model [29] contains two parts: the relational schema representing the structural and organizational information in forms of class, binary relations, relation chains and properties, and the probabilistic models which annotate the conditional probabilistic dependency relationship between properties of classes. The probabilistic model introduces the concept of p-classes. P-classes are similar to usual classes, but include special class properties, which contain probabilistic information, called p-properties. A p-property may have several restrictions: hasValue (expresses the list of possible values for the p-property), hasPD (specifies the probability distribution over those values), hasParents (specifies a list of property-chains on which the value of the property depends), hasCPT (specifies the conditional probability distribution over the values of the property given the values of its parents). The Unified Context Ontology [29] is also based on a layered approach including a high-level ontology which captures the general knowledge in the domain and a set of domain-specific ontologies, which defines specific knowledge in each domain of interest. The domain specific ontologies are composed of the relational schema and the probabilistic models, thus defining the details of each generic concept: relations, relation-chains, conditional probabilistic dependencies.

The web ontology language OWL seems to be a common choice in the most recent context-aware applications. The Unified Context Model augmented OWL with new language elements (to facilitate the representation of new concepts: relation-chain, property-chain, probabilistic dependency), a language called PROWL (probabilistic annotated OWL). By supporting a probabilistic relational model, the Unified Context Model supports not only the basic reasoning types (rule-based reasoning and ontological reasoning), but also Bayesian reasoning. An important characteristic is that they also provide an automatically mapping module, which derives a Bayesian Network from the corresponding ontologies.

Another attempt to include Bayesian Networks to support uncertain ontology representation and reasoning is suggested in [27]. Probabilities are attached to concepts and properties by extending OWL, to provide additional probabilistic language constructs. It also defines rules to derive the Bayesian Network DAG (Direct Acyclic Graph) from the OWL representation. Conditional Probability Tables (CPT's) are then attached to each node in the resulted DAG. Due to this representation of the uncertain knowledge, by means of augmented ontologies with new probability markups, General Bayesian Network inference procedures can be applied (belief propagation, junction tree). This enables the computation of the overlapping or inclusion between a concept C and a concept represented by a description e, P(C|E) [27]. The goal of this project is to develop a methodology to automatically convert from OWL ontology to a Bayesian Network.

A Bayesian Network is a tempting approach due to its expressive power and efficient probabilistic reasoning capability and also due to the similarity between the DAG representation of a Bayesian Network and the RDF graph of the OWL ontology. Still this approach raises a number of difficulties related to the method of encoding the probability values in OWL, generating a correct representation of the Bayesian Network, and the problem of constructing the conditional probability tables from user provided probabilities and logical relations specified in OWL [27].

To extend the ontology with uncertainty information, the model proposed treats the concept of probability as a resource, and defines three kinds of OWL classes: PriorProbObj, CondProbObjT, and CondProbObjF. A probability of the form P(A) (probability that an individual belongs to class A) is defined as an instance of class PriorProbObj, which has two mandatory properties: hasVariable and hasProbValue. A probability of the form P(A|B) (probability that an individual of class B also belongs to A) is defined as an instance of class CondProbObjT and a probability of the form $P(A|\overline{B})$ (the probability that an individual in the complement set of B belongs to A) is defined as an instance of class CondProbObjF, both have three mandatory properties: hasCondition, hasVariable, and hasProbValue. The properties hasCondition and hasVariable are object properties and hasProbValue is a data type property.

This approach is far from being a complete prototype; the remaining issues which are the subject of future research are the following:

- How to construct CPT for all nodes in a more systematic and disciplined way;
- Whether existing BN inference algorithm can be directly applied to this framework, and if not, what new algorithms need to be developed;
- In OWL ontologies, cycles can be formed by equivalent classes (defined by the ”owl:equivalentClass” constructor), mutual disjoint among a set of classes, or other inferred dependencies among a set of classes. Since cycles are not allowed in Bayesian networks, how to detect and remove cycles is another issue to be addressed.

Another attempt to build an environment which is able to adapt rationally to the user desire, behavior, habit or preference, is presented in [31].

The authors try to address ubiquitous computing greatest challenges, such as the system ability to learn and to predict the user preferences, in a multi-user environment.

In order to decide among the preferences of several users, the system uses the user priority concept. Priority can be fixed, but normally it changes over time, or based on situation; for example a sleeping user will be considered more important than others.

The innovative approach, presented in [31] is based on the Bayesian RN-Metanetwork, a multilevel Bayesian network used to model user priority and specific preferences. A Bayesian RN-Metanetwork is a set of Bayesian networks, laid on two levels so that the distribution of probabilistic networks on first level depends on the local probability distributions associated with the nodes of the second level network. Among the key features of the system, the authors mention the following ones:

- Bayesian RN-Metanetwork supports multi-agent systems. Each set of Bayesian networks in the first level is hold by an agent. Each agent uses the distribution of its Bayesian networks to calculate some needed values, and they communicate with other agents through some interfaces;
- The adaptation algorithm for Bayesian RN-Metanetwork is fully described to adapt the model to the continuously changing preference of users;
- Irrespective of the number of users in the environment, the priority and preference of each user is calculated separately. This is very useful for widening the scale of systems, as well as knowledge reuse.

In a ubiquitous application, the domain knowledge and user-defined rules are not enough in order to provide an online intelligent adaptation to the situation change and the user’s needs or preferences. The approach presented in [31], includes the mechanisms to address the crucial tasks of online learning and adaptation.

3.2. Context-aware ontology design and implementation

3.2.1. Ontology design

The Knowledge Engineering deals with the structuring, development and usage of Knowledge Bases. At the moment, the ontology paradigm is the most common Knowledge Engineering approach. The support for ontologies is provided by various methodologies (Diligent [67], HCOME [68], OTK methodology [69]). Uschold's methodology [71] suggests the following steps:

- Identification of the ontology's purpose;
- Capture key concepts & relationships in unambiguous textual form;
- Mapping onto precise terminology;
- Coding in formal KR language, integrating with existing ontologies;
- Evaluation and documentation for modification and reuse.

Methontology [70] is an ontology engineering methodology, for building ontologies either from scratch, reusing other ontologies as they are, or by a process of re-engineering.

The framework enables the construction of ontologies at conceptual level. The framework consists of identification of the ontology development process with the identification of the main activities: evaluation, configuration, management, conceptualization, integration and implementation. The methodology specifies the steps for performing the activities, the techniques used, the outcomes and their evaluation.

The general procedure of developing an ontology is presented in [59]. The authors stress the fact that the process of ontology development is an iterative process and that there are always alternatives solutions when modeling a domain, and the best solution is always application dependent. This methodology comprises seven main steps:

Step 1: **Determination the domain and scope of the ontology** by answering the following key questions:

- What is the domain that the ontology will cover?
- What are we going to use the ontology for?
- For what types of questions the information in the ontology should provide answers?
- Who will use and maintain the ontology?
- Competency questions: list of questions that a knowledge base based on the ontology should be able to answer [73]

Step 2: Reusing existing ontologies

Reusing existing ontologies is an important decision if our system needs to interact with other applications that have already committed to particular ontologies. Ontologies can be imported into an ontology-development environment. There are libraries of reusable ontologies on the Web and in the literature. For example, we can use the Ontolingua ontology library (http://www.ksl.stanford.edu/software/ontolingua/) or the DAML ontology library (http://www.daml.org/ontologies/). There are also a number of publicly available commercial ontologies (e.g., UNSPSC (www.unspsc.org), RosettaNet (www.rosettanet.org), DMOZ (www.dmoz.org)).

Step 3: **Definition of important terms in the ontology**

Before developing the class level is mandatory to decide on the most important concepts in the ontology and their characteristics, without worrying about overlap between concepts they represent, relations among the terms, or any properties that the concepts may have, or whether the concepts are classes or slots.

Step 4: **Defining the classes, properties and the class hierarchy**

This step is one of the most important steps in the ontology-design process.

There are several possible approaches in developing a class hierarchy [74]:

- A top-down development process starts with the definition of the most general concepts in the domain and subsequent specialization of the concepts.
- A bottom-up development process starts with the definition of the most specific classes, the leaves of the hierarchy, with subsequent grouping of these classes into more general concepts.
- A combination development process is a combination of the top-down and bottom-up approaches: We define the more salient concepts first and then generalize and specialize them appropriately.

The alternative to take depends strongly on the developers' personal view of the domain.

Classes are chosen from the list created in step number 3, out of which are selected the terms that describe objects having independent existence rather than terms that describe these objects. The classes are organize into a hierarchical taxonomy by asking if by being an instance of one class, the object will necessarily be an instance of some other class.

Once some of the classes are defined, the internal structure of concepts must be described. Out of the remaining terms from the list created at step 3, most are likely to be properties of these classes. For each property in the list, one must determine which class it describes. These properties become slots attached to classes.

Step 6: **Define the facets of the slots**

Slots can have different facets describing the value type, allowed values, the number of the values (cardinality), and other features of the values the slot can take.

Step 7: **Create instances**

The last step is creating individual instances of classes in the hierarchy. Defining an individual instance of a class requires choosing a class, creating an individual instance of that class, and then filling in the slot values.

Even if the ontology describes very specifically the steps to be followed in order to develop an ontology, the author stresses the fact that there is no single correct ontology for any domain, the quality of the designed ontology can be finally assessed only by using it in the applications for which it was designed.

3.2.2. The Protégé framework

The **Protégé** framework developed by Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine represents complete ontology and knowledge base management framework, providing the facility to integrate a wide variety of third party plug-ins.

Protégé provides two editors for ontology modeling. The Protégé-Frames editor facilitates the development of frame-based ontologies, based on a knowledge model which is compatible with the Open Knowledge Base Connectivity protocol (OKBC). This editor provides customizable user interfaces for knowledge modeling in domain-friendly forms, an extendable plug-in architecture and a wide set of additional support tools for ontology management, ontology visualization, inference and reasoning. The Java-based Application Programming Interface (API) included in the Protégé-Frames module, provides the ability for plug-ins and other applications to access and display ontologies created with Protégé-Frames.

The Protégé-OWL editor makes possible the development of ontologies using the W3C's Web Ontology Language (OWL), being fully integrated with the Jena framework.

The Protégé-OWL editor offers the necessary tools in order to use OWL and RDF ontologies, edit and visualize classes, properties and SWRL rules, define logical class characteristics as OWL expressions.

The Protégé platform provides the flexibility of integrating additional modules. Protégé supports three types of plug-ins: storage plug-ins, slot widgets, and tabs.

A storage plug-in is a module that saves and loads models in a certain file or database format (CLIPS, XML, XML Schema, RDF, OIL, DAML+OIL, DARPA Agent Markup Language+OIL, UML, XMI metamodels).

Slot widgets are graphical components such placed in Protégé's instance forms to view and edit a slot value. Protégé's plug-in library provides additional slot widgets for specific data types such as calendar and date widgets, and components that display images, sounds, and videos.

Tabs plug-ins are graphical plug-ins, displayed as a tab in Protégé's main window:

- **Visualization tabs**
  - **Jambalaya** provides a hierarchical ontology browser that allows for interactive editing of existing data;
  - **TGVizTab** provides an interactive graphs-based visualization method for classes;
  - **OntoViz** provides a configurable graphical display of models in graphs similar to UML diagrams.
- **Project and file management tabs**
  - **BeanGenerator** enforces the interaction between Java and Protégé, by generating JavaBeans classes from a Protégé class model;
  - **DataGenie** enforces the interaction between databases and Protégé, allowing Protégé access to databases using the Java Database Connectivity interface;
  - **Prompt** allows managing different domain models in Protégé (merge models, to extract a part of a model, or to identify differences between a model's two versions).
- **Reasoning tabs**
  - **Query tab** for querying the knowledge base;
  - **PAL constraint and query tabs** provide the mechanisms for editing and evaluating expressions in the PAL;
  - **JessTab** connects Protégé to the Java Expert System Shell (JESS).

Protégé uses a declarative approach to model ontologies, declaring explicitly the class hierarchies and individual membership to classes.

The main elements of ontology are: classes (also called concepts), concept features and attributes (called slots, roles or properties), and restrictions on slots (facets, role restrictions).

The ontology concept has common elements with the object oriented design approach.

Protégé classes are similar to Java classes, can be arranged in an inheritance hierarchy, but they do not provide method definition as in Java. Classes can be defined as abstract or concrete, only the latter can be instantiated. Protégé also supports multiple inheritance.

Attributes (slots), have a name and a value type: primitive value types: (boolean, integer, float, or string), symbols (represent enumerations of strings), or references to the instances and classes. Slots are also uses to build relationships between instances.

Although slots are similar with the class attributes in object oriented languages, there are some specific features of Protégé slots:

- a slot can attach to multiple classes;
- slots can have values constraints;
- slots are global objects (they can even exist without being assigned to a class); for each slot's assigned class, the developer can keep the general properties or override the slots' properties;
- Protégé Axiom Language (PAL), is a very powerful Protégé built-in language, which can be applied when dealing with more complex constraints;
- The JessTab extension can be employed to express complex constraints.

3.2.3. Protégé integration with JESS

Jess (Java Expert System Shell) is a Java-based rule engine, consisting of a rule base, fact base, and an execution engine. The JessTab provides rule based reasoning mechanism over Protégé ontology. JessTab allows the development of ontologies in Protégé and running problem solvers in Jess that use the Protégé knowledge bases. It provides the facility to map Protégé knowledge bases to Jess facts, and allows the modification of the Protégé knowledge base by class instantiation or attribute value change. JessTab includes editors for creating or redefining Jess rules and functions.

Jess tab provides the following key features:

- Jess console window in Protégé;
- Mapping instances to facts;
- Functions for knowledge-base operations;
- Mirroring Jess definitions in Protégé knowledge bases;
- Support for meta-level objects.

Because JESS requires explicit definitions of the data types in the form of templates, JessTab provides the mapping command in JessTab which builds these templates based on the classes and slots in the ontology. For reasoning over the knowledge base, JESS requires a set of facts (instantiations of the templates), which does the JessTab mapping command provide.

Among the strength featured by the Jess system, is the fact that it can be naturally integrated with other systems, being one of the most flexible rule engines at the moment.

Beside the integration with Protégé, Jess has been integrated with agent frameworks and other tools, including the JADE framework.

3.3. Agents development methodologies

3.3.1. Software development methodologies

Software development provides guidelines for the system designer when developing a system. As agents become widely recognized as a promising technology there is a growing need for practical methods for developing agent applications. Several attempts for creating tools and methodologies for building multi agent systems have been developed recently. Still most of these approaches focus on specific architectures and thus lack the support for a wider range of agent architectures or do not support highly complex system development.

Among the most recent agent-oriented software development methodologies, are Gaia (Zambonelli, 2003), Prometeus (Lin Padgham and Michael Winikoff, 2002), MaSE (Wood and DeLoach, 2001), Tropos (Mylopoulos, Pistore, Kolp, 2000), Adelfe (Bernon, Gleizes, Peyruqueou, Picard, 2003), PASSI (Cossentino, Sabatucci, 2002). Unfortunately, they cover only the requirements (MaSE), analysis, and design phases (MaSE, Gaia) of the software development cycle (Sommerville, 2000).

There have been developed frameworks to support both methodology and the agents' development environment, like Zeus (Collis and Ndumu 1999) or Agent-Tool with MaSE (DeLoach and Wood 2001).

In [37] a combined approach is proposed, which aims to implement a multiagent system with the JADE framework using the Gaia methodology for analysis and design. Here the process is basically dedicated to the conversion of Gaia models to JADE code. This framework covers the full software development lifecycle, the conversion process being described using the Software Process Engineering Metamodel (SPEM).

The Gaia methodology is focused on the analysis and design of Multi Agent Systems. Gaia defines Multi Agent Systems as an agent society in which autonomous agents interact and play some roles. Gaia uses the notion of role model to identify the roles that agents have to play within the MAS and the interaction protocols between the different roles [37]. The Gaia modeling methodology represents a three-phase process: the analysis phase, the architectural design phase and, finally, the detailed design phase. The analysis phase requires the identification of the roles (having the role attributes: responsibilities, permissions, activities, and protocols) and the modeling of interactions between the roles. The responsibility attribute is an important aspect of the role, defining the functionality. Permissions attribute defines the permission access to information resources. The activities are tasks that an agent performs individually, not by interacting with the rest of the agents. The protocols attribute represents the specific patterns of interaction.

During the analysis phase other steps have to be performed: the identification of the possible interactions with a role’s external environment and defining the global behavior organizational rules.

In the architectural design phase, the roles and interactions models augmented by the definition of the system’s organizational structure in terms of its topology and control regime.

Finally, the detailed design phase maps roles into agent types and specifies the right number of agent instances for each type. In this manner, an agent type is an aggregation of one or more agent roles. According to Gaia, the service model represents the services fulfilled by the roles in one or several agents.

According to the specifications in [37], the JADE implementation phase can be described as a four-step process:

1. Communication protocol definition, meaning the definition of all the necessary ACL messages together with the definition of the possible ordering of their exchange.

2. Activities refinement table definition, specifying application-dependent data, their structure and the algorithms that are going to be used by the agents.

3. JADE behaviors definition.

4. Creating the agent classes.

The Gaia2Jade overall development process is top-down in the analysis and design phase (i.e., by using Gaia) and bottom-up in the implementation phase, according to the most successful software engineering practices.

The Gaia2Jade process results showed that Gaia methodology is robust and reliable, and the produced models and schemata were used throughout the project development phases as a reference. It proved to be flexible enough, so that it was easy to iterate through the design and implementation phases, as is demanded by modern information systems development [37].

3.3.1.1 The Tropos methodology

The Tropos methodology provides a modeling language based on a multi-agent paradigm; it supports analysis techniques and the system design process.

The Tropos modeling language allows the representation of intentional and social concepts, such as actor, goal and soft-goal, plan, resource, and a set of relationships between them, such as actor dependency, goal or plan decomposition, means-end and contribution relationships. The modeling process starts with the identification of critical actors in a domain along with their goals, and proceeds with the analysis of goals on behalf of each individual actor. In particular, given a root goal of an actor, the software engineers may decide to delegate it to an actor already existing in the domain or to a new actor. Such delegations result in a network of relationships among actors in the domain. Moreover the software engineer may decide to analyze a goal producing a set of sub-goals. Goal analysis generates a goal hierarchy where the leaves in various combinations represent concrete solutions to the root goal. Finally the software engineer may decide that a certain actor is able to satisfy the goal via a plan the actor is able to execute. In this case the goal is assigned to that actor (with no further delegations). The process is complete when all goals have been dealt with.

The development process in Tropos consists in five phases: Early Requirements, Late Requirements, Architectural Design, Detailed Design and Implementation.

The **Early Requirements** phase identifies actors and goals represented by two different models. The actor diagram depicts involved roles and their relationships, called dependencies. These dependencies show how actors depend on each other to accomplish their goals, to execute their plans, and to supply their resources. The goal diagram shows the analysis of goals and plans regarding a specific actor in charge of achieving them. Goals and plans are analyzed based upon reasoning techniques such as means-end analysis, AND/OR decomposition, and contribution analysis.

These models will be extended in the **Late Requirements Phase**, which models the systems within its environment. The goals can be decomposed into sub-goals. The **Architectural Design** has the objective to specify the system architecture in terms of a set of interacting software agents.

The **Detailed Design** phase deals with the detailed specification of the agents' goals, belief and capabilities. Also communication among agents is specified in detail. This phase is usually strictly related to implementation choices since it is proposed within specific development platforms, and depends on the features of the adopted agent programming language. The **Implementation** phase has the objective of generating code from the detailed design specification.

3.3.1.2 TAOME- A Design Framework for Generating BDI-Agents

The TAOM4E (Tool for Agent Oriented Visual Modeling for the Eclipse platform) tool supports the Tropos agent-oriented software engineering methodology. This framework was developed in order to provide the CASE tools for a consistent, powerful and customizable software development process. TAOM4E is a plug-in for the Eclipse platform, which implements the whole Tropos metamodel.

Tropos actor diagrams can be graphically created and extended for early and late requirements analysis phases. Each actor can be detailed in a goal diagram, shown in a “balloon”, where delegated goals are displayed and can be decomposed. System actors can then be detailed to a multi-agent system, in architectural design diagrams.

TAOM4e embeds an Agent Code Generation Tool, which performs the conversion of the transformation of the Tropos model specification into Jade code.

3.3.2. Agent prototyping environments

With the emergence of the agent prototyping environments which provide visual tools and interfaces to aid the design and realization of diverse agent based systems, the efforts for the developing such systems is considerably reduced. Agent prototyping environments provide the tools for modeling, integrating and developing of software agents.

Because of the wide spreading and continuous development and maturing of an extremely large industrial offer of toolkits and platforms on agent design, an evaluation of these systems needs to be approached. Generally, before choosing the most appropriate platform for developing multi-agent systems, a thorough analysis of the system requirements, followed by the assessment of these requirements on the latest version of the available agent prototyping environments, is strongly encouraged. Of course, each approach has its own strengths and weaknesses in the support it provides for different aspects of agency and for other aspects such as popularity, availability, maintenance and documentation.

3.3.2.1. Agent platforms evaluation

Agent platforms provide an environment for the execution of agents and often facilities and services to manage agents on the platform, for example life-cycle (starting, stopping, resuming deleting agents) and communication facilities and Directory Services

(White and Yellow Pages). Additionally agent platforms can offer support for migration and security.

Several agent prototyping environments comparative evaluations exist in the literature.

In [40] the analysis of agent platforms is based on the following of key features: compliant standards, communication facilities, openness, availability, mobility, security and the documentation support. This technical report focuses on the following platforms: Aglets Software Development Kit, Ajanta, Tryllian’s Agent Development Kit, FIPA-OS, JADE, JACK Intelligent Agent and Zeus. The candidate platforms according to [40] are JADE and Aglets, due to their high popularity among developers, very good documentation, high acceptance of users, very good security features, standard commitment (FIPA or MASIF), good Graphical User Interface, and diversity of communication protocols. Other platforms were either not compliant with standards (Ajanta), have poor Graphical user interface (Ajanta), have no mobility support (Zeus, JACK), or the commercial version was very expensive (Tryllian).

The agent platform evaluation provided in [54] stresses the importance of updating the evaluation when developing a new project. One cannot rely on previous evaluations, as most of the platforms were released by academic environments or companies’ laboratories for research purposes, and they may not be available or maintained. Based on this idea, the evaluation criteria attempt to answer the following questions:

- Is the platform still maintained?
- Is the platform’s authors’ research group still active?
- Is the platform being developed?
- Is the platform popular? Is it in broad use?
- Is the platform easy accessible?
- What is the date of the latest release of platform?
- Does a light-weighted release of the platform exist?
- How is the platform available?

The eight platforms evaluated were Agent Development Kit (Tryllian), April Agent Platform, Comtec Agent Platform, FIPA-OS, JACK Intelligent Agents, JADE, JAS, ZEUS. The evaluation results indicate that the only platforms which satisfy these criteria are ADK, JACK and JADE. JADE differentiates itself from the other two candidate agent platforms, as it is the only free platform (Open Source, GNU General Public License) and it is also very well documented. It is also a very popular platform, being widely accessible, and continuously developed, improved and maintained, not only by the developers from Tilab, but also by JADE community members. According to the evaluation in [54], JADE is the best option since it supports the development of ontologies which can be designed using Protégé and then adapted to JADE using JadeJessProtege plugin.

Another interesting approach for evaluation of agent platforms in ubiquitous systems is provided in [55]. The paper attempts to assess 19 multi-agent platforms in order to decide their appropriateness as an infrastructure for ubiquitous computing. The basic requirements are divided into two sets: the compulsory requirements (mobility, FIPA compliance), and the ranking requirements (security, heterogeneity and scalability). After the initial selection, based on the compulsory requirements, three agent platforms remained for further evaluation based on the ranking requirements: MicroFIPA-OS and Jade/LEAP. The study identified JADE/LEAP to have the better result in the ranking evaluation. According to this study, JADE/LEAP has its strength in scalability. The theoretical study and evaluation is associated with an experimental study, to assess the scalability criterion and memory consumption when using the three multi-agent platforms. The experiment revealed the MicroFIPA-OS and JADE/LEAP are scalable, when increasing the number of agents. The assessments of memory consumption, indicated for JADE/Leap values approximately 50 % lower than MicroFIPA-OS. However, the general conclusion of this study was that JADE/LEAP multi-agent platform had the highest rankings, while providing most of the needed characteristics.

The most recent attempt (2007) to classify the existing platforms is presented in [41]. The classification addresses the following mandatory requirements for the agent platform: the ability to run on mobile devices, to be an open, FIPA compliant platform, to offer the possibility to program the agents, and optionally, it should be open source, and support a representation language. The choice of possible platforms is based on available FIPA compliant platforms, proposed on the FIPA website: Tryllian, April, Comtec, FIPA-OS, JACK, JADE, JAS, ZEUS. Some of the platforms (April, Comtec, Jack, and Zeus), provide no documentation related to the requirement of running on mobile devices. Jack, Tryllian are not available as open source, while JAS is not programmable, so all these platforms fail to meet all the requirements. Although FIPA-OS is also compliant with the six criteria, the JADE platform was selected, as the developers had prior positive experiences with it, and also because this platform features a very strong and active development community.

Table 3.2. shows a comparison between four agents platforms.

Table 3.2. Agent platform evaluation

	Aglets	**Zeus**	**JADE**	**JACK**
Compliant standards	MASIF	FIPA	FIPA	FIPA
Communication	sockets, message-passing between agents, ATP (support HTTP tunneling)	KQML and ACL	ACL, support for inter-platform messaging with plug-in MTPs (RMI, IIOP are ready and HTTP, WAP are already scheduled), ACL and XML codec for messages	needs DCI network for communication; similar to TCP/IP it needs one process running as a name-server
Mobility	weak	no	weak	no
Security	build-in security mechanism, context and server security	ASCII-encoded, Safe-Tcl scripts or MIME-compatible e-mail messages for transportation; using public-key and private-key digital signature technology for authentication	connection authentication, user validation and RPC message encryption - socket proxy agent acting as a bidirectional gateway between a JADE platform and an ordinary TCP/IP connection	internal security provided by JDK
Availability	free software -binary and the open source are available	free software	free software, GNU public license	evaluation version is free
Documentation	good	not too well documented	very well documented	no documentation for mobile devices employment
Open source	yes	yes	yes	no
Is the platform still maintained	rather not, last release: 2002	yes, last release January 2006	yes, last release June 2007	yes

3.3.2.2. The Agent Factory

One of the most promising attempts to build up a cohesive framework for the development and deployment of agent-based systems is the Agent Factory. It gives support for the creation of Belief-Desire-Intention (BDI) agents [33].

Agent Factory is structured into two main environments: the Agent Factory Development Environment (which provides Computer-Aided Software Engineering tools to support the agent fabrication process), and the Agent Factory Run-Time Environment (delivers support for the agent deployment over a domain of Java-compliant devices).

The core layer exposes the necessary functionality to develop BDI agents and to facilitate agent communication based on ACL (Agent Communication Language). The layers above assist the developer in the phases of design, implementation, deployment (Agent Factory Development Environment), and the appropriate methodology for building such systems (Agent Fabrication Process). The Agent Fabrication Process methodology stands out among the agent methodologies by the fact that it provides consistent support for all project development phases, except the analysis phase.

Support for the Agent Fabrication Process AFP is realized through an integrated toolset known as the Agent Factory Development Environment.

The Agent Fabrication Process provides rich tools for Ontology definition (Ontology Editor), Actuator and Perceptor Building (Actuator Builder tool, Perceptor Builder tool), Agent Coding (Agent Design Tool), Agent Testing (Agent Viewer Tool, Community Monitor, and Message Sender), Application Deployment (Script Builder tool). A central feature to the Agent Factory architecture is the construction of agents from various components. Each agent comprises a set of components and a partial agent program, representing an instantiation of the agent design. According to [33], the agent design represents a class hierarchy, more specifically aggregation of pre-existing modules and protocols that are joined through glue code. The agents created in the Agent Factory are BDI (Belief, Desire and Intention) agents, created by implementing mental state architecture. The agents have the possibility to reason using an interpreter, which reasons based on the agent's mental state.

The Agent Factory Agent Platform provides the functionality for agent deployment and it comprises the Message Transport System (which controls the mission transmission towards the agents on other platforms), and other three components: a White Pages Agent (manages the creation and deletion of agents on the Agent Platform), a Migration Manager Agent (supporting only weak agent migration through cloning) and also may contain a

Yellow Pages Agent, the equivalent of Directory Facilitator in the FIPA specification, providing the agents information on currently registered services.

The Agent Fabrication Platform has been proved to be a viable and robust agent development system, being successfully used in various applications such as: Ad-me [44], Gulliver's Genie [45] and EasiShop [46]. These applications share a set of common features: they use the Agent Factory platform, they are designed to work and adapt in dynamic, rapidly changing environments. The agents in these systems are implemented in Java, as context sensitive applications.

Ad-me (Advertising for the Mobile E-commerce user) provides intelligent context sensitive tourist services, aiming at providing advertisements to users based on their location and preferences. In order to model the dynamics of these mental attitudes, Ad-me system is built upon BDI architecture.

Gulliver's Genie is based on the idea of providing users an electronic tourist guide. This context aware system also adopts a multi agent paradigm, implementing mobile intelligent agents from the BDI family. Gulliver's Genie aims to provide an interactive, intuitive and dynamically adapting medium, to both the mental state of the tourists and to their spatial context. To maximize portability, they also used Java as application programming language. Gulliver's Genie employs standard client/server architecture. The client is hosted on a PDA, with GPS and GPRS facilities, is comprised of several components:

- the Spatial Agent (extracts both position and orientation readings from the GPS sensor);
- the Cache Agent (manages the cache on the client, ensuring correct information based on to the tourist's immediate location and in accordance to any recent changes to his interest profile);
- the PDA Controller (for initializing and terminating Genie sessions);
- the Interface Handler (manages the user interface);
- the Communications Handler (schedules the dispatch of messages to the server).

The Agent server architecture is based on the following agents:

- the Registration Agent (assigns an individual Agent from the Tourist Agent template to the user);
- the Profile Agent (updates the tourist's profile);
- the GIS Agent (handles cache update requests from the client);
- the Tourist Agent (acts as an interface between the client and the server);

- the Presentation Agent (builds and caches presentations for all attractions in the tourist's immediate vicinity).

One of Genie's key features is its proactive anticipation of user preferences and the intelligent pre-caching of presentation content on the tourist's local device, thus accomplishing a real-time distribution of information to users.

EasiShop, a context aware ubiquitous commerce application, enables the dissemination of context sensitive information (conditions-cheapest price, best availability, best after sales service, geographic proximity) to shoppers while walking in shopping malls. Each user is registered with a shopping agent who will provide guidance and assistance in the shopping process. This is achieved through registering user profiles at the system initiation, and then dynamically adapting to the changes in the shopper's preferences or intentions. The EasiShop agents can migrate to a market place and participate in an auction by adopting a Contact Net Protocol. Each shop is represented by agents responsible with the sale of their products. The purchase process relies on the negotiation and trade between the PDA Agent and the representative Store Agent.

Systems such as Ad-me, EasiShop or Gulliver's Genie provide evidence for the feasibility of supporting mobile intentional agents for the delivery of location-aware and context-aware services.

3.3.2.3. The JADE platform

JADE is the abbreviation for the **Ja**va Agent **DE**velopment Framework and was developed to be compliant with the FIPA specifications for intelligent agents [53]. JADE is a middleware that simplifies the development process of Multi Agent Systems by providing a set of agent technologies that support the debugging and deployment phases. It provides the following services specific to distributed multi-agent applications:

- agent life-cycle and agent mobility;
- white and yellow-page services;
- peer-to-peer message transport and parsing;
- distributed security, fault tolerance, support for replicated agents and services, persistence;
- multiple tasks scheduling;
- application-specific persistent delivery filters;

- set of graphical tools to support monitoring, control and debugging (RMA (Remote Monitoring Agent), Dummy Agent, Sniffer Agent, Introspector Agent, Log Manager Agent, DF (Directory Facilitator) GUI).

The JADE agent platform can be distributed on several hosts, regardless the underlying operation system and it also provides efficient communication of ACL (Agent Communication Language) messages. The communication model is based on asynchronous message passing. JADE also provides methods for message filtering. The developer can apply advanced filters on the fields of the incoming messages. Message polling can be blocking or non-blocking with an optional timeout.

Extending the JADE agent class allows creating new agents. In this way new behaviors can be defined, and also the agent's initialization and termination functionality. By using the GUIAgent class, agents with a graphical user interface can be created, allowing for the agent's interaction with a human user. Each Agent instance is identified by an AID (jade.core.AID) which comprises a unique name and an address. On a JADE platform agents are referred only through their names.

In JADE, agent tasks are realized through the use of behaviors. Behaviors are created by extending the jade.core.behaviours.Behaviour class. Behaviors are logical execution threads that can be composed in various ways to achieve complex execution patterns and can be initialized, suspended, and spawned at any given time. An agent can have several behaviors executing in parallel, but everything occurs within a single Java Thread. When there is no active behavior for an agent, it enters the idle state and its thread goes to sleep.

The agent core keeps a task list that contains the active behaviors. A scheduler carries out a round robin policy among all behaviors available in the queue. Each behavior performs its designated operation by executing the core method action(). Behavior is the root class of the behavior.

Beyond the core features, JADE also provides advanced features [43]:

- Interaction Protocols, predefined sequences of messages exchanged by agents during conversations;
- AMS (Agent Management System) represents controls platform management operations (creating/killing agents, killing containers, etc.);
- Mobility of status - an agent can move to a remote container (on a different host) and restart its execution from the exact point where it stopped on the initial container and code (if the code of the moving agent is not available on the destination container it is automatically retrieved on demand);

- Security prevents malicious actions, by using JAAS (Java Authentication and Authorization Service);
- In-process interface allows using JADE (creating a container and starting agents) from an external Java program;
- Threaded behaviors allow executing a normal JADE behavior in a dedicated thread;
- Persistence (allows saving and reloading agent state on relational database).

One of the essential requirements when dealing with multi agent systems is to have the ability to answer queries by using facts and rules stated in an expert system. The Expert System is the part of the application that will control the behavior. An expert system shell includes a knowledge base and inference engine. The Expert System must be rule-based, portable, and free or open source.

At the moment, Jess is one of the fastest and most powerful Java based rule engine and scripting environment. In the multi-agent systems world, it can be used as a decision component of an agent. Jess has many outstanding features including backwards chaining and working memory queries, the ability to directly manipulate and reason about Java objects. [52]. Jess is also a scripting environment, granting facilities to create Java objects, call Java methods, and implement Java interfaces without compiling any Java code.

JADE allows full integration with Jess, offering the so-called “JessBehaviour”.

JADE supports ontologies created by standard tools such as Protege-2000 through what the JADE team calls schemas. Schemas are Java classes designed to represent the static structure of ontology. This structure is supported by either Java classes or abstract descriptors associated with each schema. Element schemas are objects describing the structure of concepts, actions, and predicates allowed in messages.

JADE provides a sophisticated mechanism, to handle ontological elements as instances of Java. When moving from the design to the implementation, the creation of these ontological Java classes is very straightforward, but, especially when dealing with large ontologies with a lot of templates, it may be quite time consuming. Another approach to integrate ontologies is by using the “beangenerator” plug-in, by which it is automatically created the ontology definition class and the predicates, agent actions, and concept classes, from the ontology specified in Protégé.

3.3.2.4. Jadex- agent-oriented reasoning engine

Jadex (JADE eXtension) is a software framework built on top of the Jade platform, which supports the implementation of Jade agents, which exhibit a rational, goal-oriented behavior.

The framework is designed for easy integration into JADE as an add-on package, and consists of an API, an agent platform and development tools. Jadex is an implementation of hybrid (reactive and deliberative) agent architecture for representing mental states in JADE agents following the BDI model. In Jadex, the reasoning engine is clearly separated from the underlying architecture.

Its' objective is to build up a rational agent layer that sits on top of a middleware agent infrastructure and allows for intelligent agent construction using sound software engineering foundations. Furthermore, Jadex – Jade approach in the development of rational agents combines the strength of the well tested agent middleware with the benefits of the abstract BDI execution model.

3.3.2.4.1. Jadex Architecture

Jadex reasoning engine's architecture has the roots in the PRS computational model. Jadex provides the tools for the development of agents with explicit representation of mental attributes (beliefs, goals and plans), that automatically deliberate about their goals and selecting the appropriate plans to achieve their goals.

As illustrated in fig. 3.2., an agent is a black box, which receives, processes the received message and sends the response messages. Incoming messages, as well as internal events and new goals serve as input to the agent's internal reaction and deliberation mechanism. Based on the results of the deliberation process these events are dispatched to already running plans, or to new plans instantiated from the plan library. Running plans may access and modify the belief base, send messages to other agents, create new top-level or sub-goals, and cause internal events.

An agent is described in an XML file, called agent definition file (ADF). The agent is specified by declaring beliefs, goals, available plans and some other properties. In addition to this, for each plan used by the agent the plan body has to be implemented in a separate Java class.

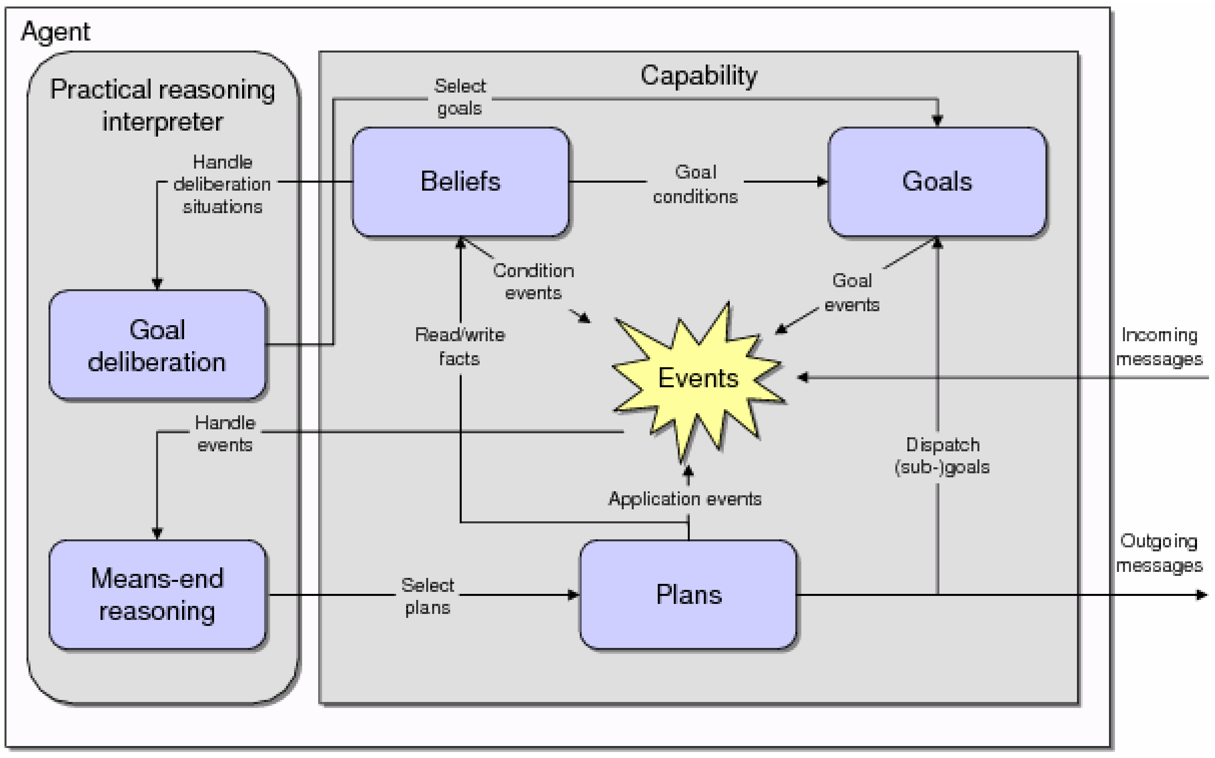


Fig. 3.2. Jadex Abstract Architecture

Beliefs

Each Jadex agent stores its knowledge in a special structure, called Belief Base.

In Jadex there is no need for special constructs for knowledge representation, Java objects can be stored as facts or by using knowledge modeling tools like Protégé and code generators, it is possible to capture the semantics of objects, which are stored as beliefs.

The belief base implementation also integrates concepts from relational databases, by providing a set oriented query language, which allows retrieving subsets of beliefs, or evaluating expressions over the belief base state. Additionally, Jadex allows the specification of conditions, persistent expressions of a belief or set of beliefs state. Once a condition is satisfied, an internal event is generated, which may trigger plans or plan steps, or lead to the adoption of new goals.

Goals

In Jadex, goals are represented as explicit objects contained in a Goal Base, which is accessible to the reasoning component as well as to plans if they need to know or want to change the current goals of the agent. Three different kinds of goals can be adopted: achieve, maintain, and perform goal. The achieve goal just defines a desired target state, without specifying how to reach it. Agents may try several different alternatives, to achieve a goal of this kind. For goals of kind maintain, an agent keeps track of the state, and will continuously execute appropriate plans to re-establish the target state whenever needed. When not the desired target state, but rather the concrete actions to be done are the matter of subject, a goal is of kind perform. Perform goals directly specify the actions to execute, therefore an agent will not engage in any meta-level reasoning how to achieve a goal of this kind.

Reasoning about goals can be implemented in two ways. Using activation and deactivation conditions, goals can be enabled or disabled when certain internal conditions hold. Alternatively, goals maybe activated and deactivated manually from the implemented plans.

Plans

Jadex uses the plan-library approach to represent the plans of an agent. The agent functionality is decomposed into separate plans, which are implemented as Java classes. Therefore object-oriented techniques can be exploited in the development of plans

The Jadex execution model is event-based. Everything happening inside a Jadex agent is represented as event. Message events denote the reception of an ACL message. Goal events announce the emergence and the achievement of goals, and internal events report changes of beliefs, timeouts, or that conditions are satisfied. Events trigger plan steps, by leading to an invocation of the action method of a plan. A plan step is executed as a whole, and may contain several basic actions and/or sub-goals. Plans create filters to wait for specific events, which trigger subsequent plan steps. In addition, activation filters are used to specify which plan should be instantiated when a certain event occurs. The event model allows already running plans to wait for specific goals to appear, which are then adopted by the plan. This is a fundamental difference to traditional PRS-style plans, which are created for a single root goal, and are deleted, once they have achieved, or failed to achieve the goal. Each running plan has its own internal wait queue, which collects events for the plan to process later. When no plan is currently interested in an event, the event is ignored and discarded.

3.3.2.4.2. Jadex Development Tools

As a Jadex agent is still a JADE agent, all available tools of JADE can also be used to develop Jadex agents. Additionally Jadex embeds various tools for runtime and debugging activities as well as for development and documentation: Jadex Control Center, Starter, DF Browser, Conversation Center, Introspector, Tracer, Test Center and Jadexdoc

Jadex agents can be executed through the graphical Control Center, on a Jade platform extended by the Jadex Remote Monitoring Agent. The Jadex Introspector plug-in can be used to observe internal state of agents including their beliefs, goals and plans. The Tracer Agent can be used to visualize the internal goal achieving processes of an agent at runtime and show causal dependencies among agent's beliefs, goal, plans and message events, textually or in a graphical, molecule-like fashion. Among the development tools, Jadexdoc is worth to be mentioned. This documentation tool creates JavaDoc-like documentation from ADF files.

3.3.3. JADE based multi-agent applications

JADE is probably the most popular agent-oriented middleware used in a large range of multi-agent applications, by providing support for a run-time environment and also a set of graphical tools to support agent monitoring, control and debugging. As it grants support for multi-agent systems, JADE has been used as a development tool for applications ranging from small systems for personal assistance to large, complex systems for industrial applications.

3.3.3.1. My Smart Agenda Manager Project

JADE-LEAP was used in the **mySAM** [36] project, to support the deployment of context-aware agents, which can learn to take decisions in a dynamic, changing environment. The stated goal of this project was to develop a multi-agent system, comprising several agent societies, which are able to share their common knowledge about the context, in order to take actions based on the relevant context aspects.

The system is based on the deployment on PCs or handheld devices of several meeting scheduling context-aware agents, which give support for scheduling meetings, by negotiating with other peer agents. The architecture developed in this project is organized on four layers: context sources collecting context information from sensors, a context management layer, a reasoning layer based on agents that reason on and with context, and application layer in which context can be used directly by the application itself.

Context Managers use ontologies to handle the context information available in a certain agent society. The agents retrieve context knowledge from the Context Managers. To avoid dealing with a huge ontology, each Context Manager only manages local context ontology, having the responsibility to compute the values of the context attributes. Agents learn how to identify relevant context and have the possibility to share with other agents the way they use the context knowledge in solving similar problems, thus the agents can learn from each other’s experience. Each deals with particular domain ontology and a set of context attributes, being able to answer queries related to its managed context attributes.

The mySAM comprises several mySAM agents, deployed with the JADE/LEAP platform and a Context Manager Service, all these entities implemented as JADE agents. The context knowledge base agent contains ontology created using Protégé. Jena, a Java library designed for ontology management, facilitates the access to the ontology.

3.3.3.2. The C@sa project

Another project, in which the JADE framework was used for developing a multi-agent system, is the **C@sa** Agent-Based Home Simulation and Control [56].

This project aims at developing a context-aware pervasive system, which is able to control the house behavior according to the context information.

The MAS developed for the C@sa, is based on a layered organization of different types of agents: operators, supervisor and interactors.

The operator agent controls the behavior of devices or appliances. It is defined by a set of attributes describing the state of the device and a set of behaviors describing the task that the user or another agent can perform on it. Each task can generate natural language explanations, which can be used to assist the user when handling that device. The operator agent can have the following functionalities: to measures the value of one or more device attributes (temperature, humidity, etc.), or to directly modify the state and/or other device parameters.

The supervisor agent (e.g. Comfort Agent) ensures the coordination of the operator agents in its influence sphere with the specific user preferences and needs. An influence sphere is defined as a function of the type of service provided, for example comfort, security, wellness or entertainment. The Agent’s decisions are based on an influence diagram that models the relationship between decisions, random uncertain quantities (user goals) and values (utility of the action) (fig. 3.3.). Given a certain context configuration defined by sensor values, user goals and preferences (defined as a probability function), it can be calculated the utility of performing a certain action by the Supervisor Agent. In order to provide a model for decisional behaviors, a dynamic, uncertainty-based knowledge representation approach has been used. The Supervisor agent stores a model of the user's needs and preferences within its influence zone. When the Supervisor Agent makes a decision about a particular course of actions, it requests the Operator Agent to perform the action. The communication between the agents is based on the ACL protocol.

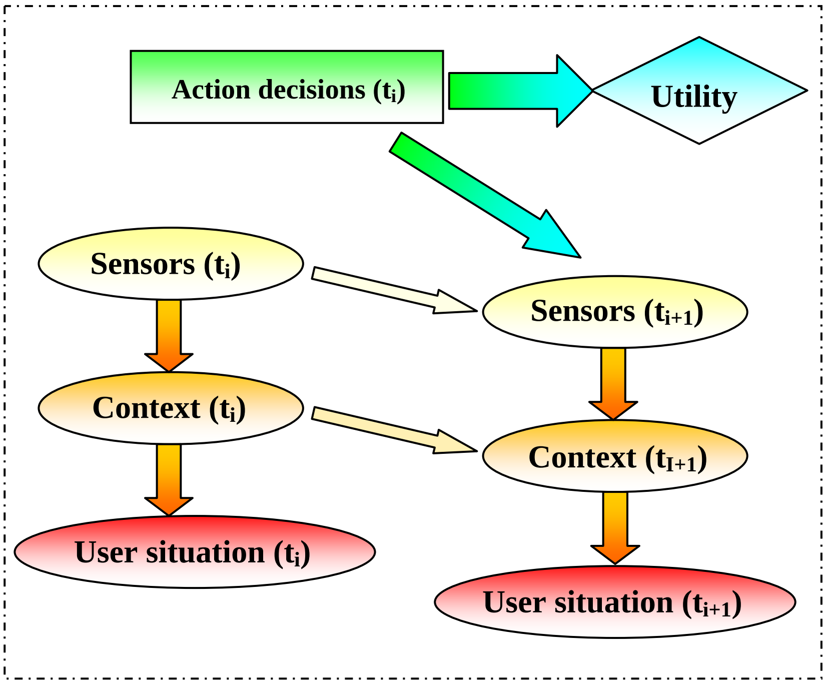


Fig. 3.3. C@sa project -A general Decision Schema of a Supervisor

3.3.3.3. The CHIL project

The **CHIL** project (Computers in the Human Interaction Loop) represents an implementation of an agent-based middleware infrastructure for pervasive ubiquitous computing for in-door environments [57]. The CHIL approach is built upon implementing fundamental issues in ubiquitous applications including the concept of context-awareness in smart spaces, the integration, management and autonomous operation of sensors, perceptual components, context modeling components, and actuating services. The system was implemented using JADE framework.

The CHIL project provides a set of perceptual (2D-visual components, 3D-visual perceptual components, acoustic components) and multimodal interface technologies. Perceptual components and multimodal interfaces are integrated into pervasive context-aware services, integrated in prototype smart spaces called “smart rooms”. Smart rooms consist of a rich collection of artifacts such as: cameras, different types of sensors and actuating devices.

The CHIL architectural framework comprises a layered approach based on three tiers (fig. 3.4.).

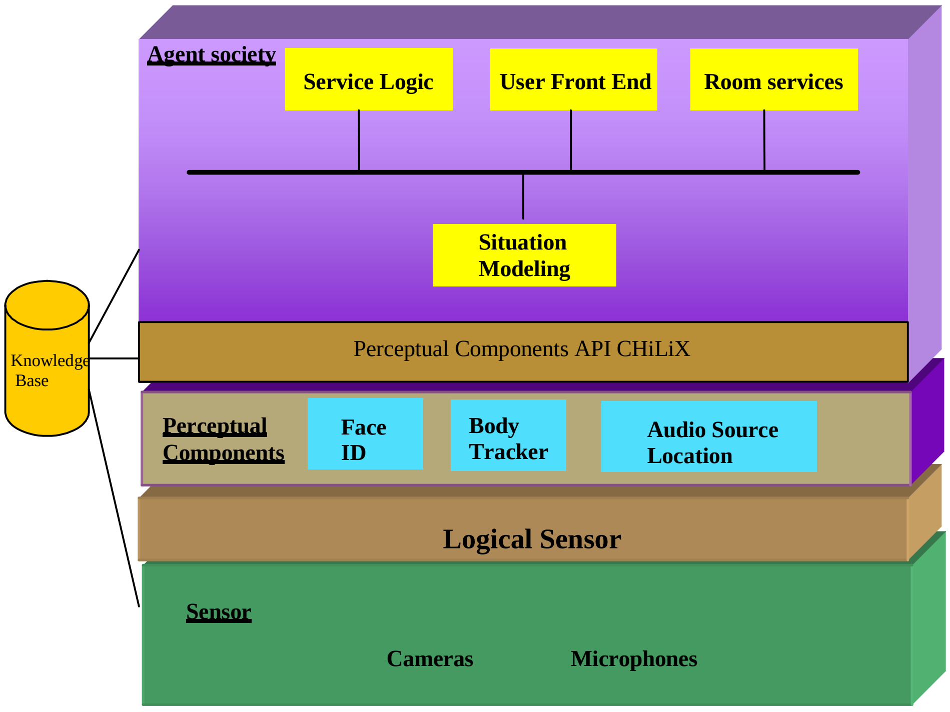


Fig. 3.4. The CHIL architectural framework

- A sensors tier used to collect information from the environment;
- A tier of perceptual components based on signal processing algorithms. Context derived from perceptual components relates to the identity and location of people and objects;
- A tier of agents that model and keep of track higher level contextual situations, while at the same time incorporating the service logic of the pervasive computing service.

The agent tier, comprising a multi-agent society is responsible for: transparent, distributed communication between components, higher level context-modeling, intelligent control and management of the actuating services, graphical presentations to terminals and devices, access control to the services (fig. 3.5.).

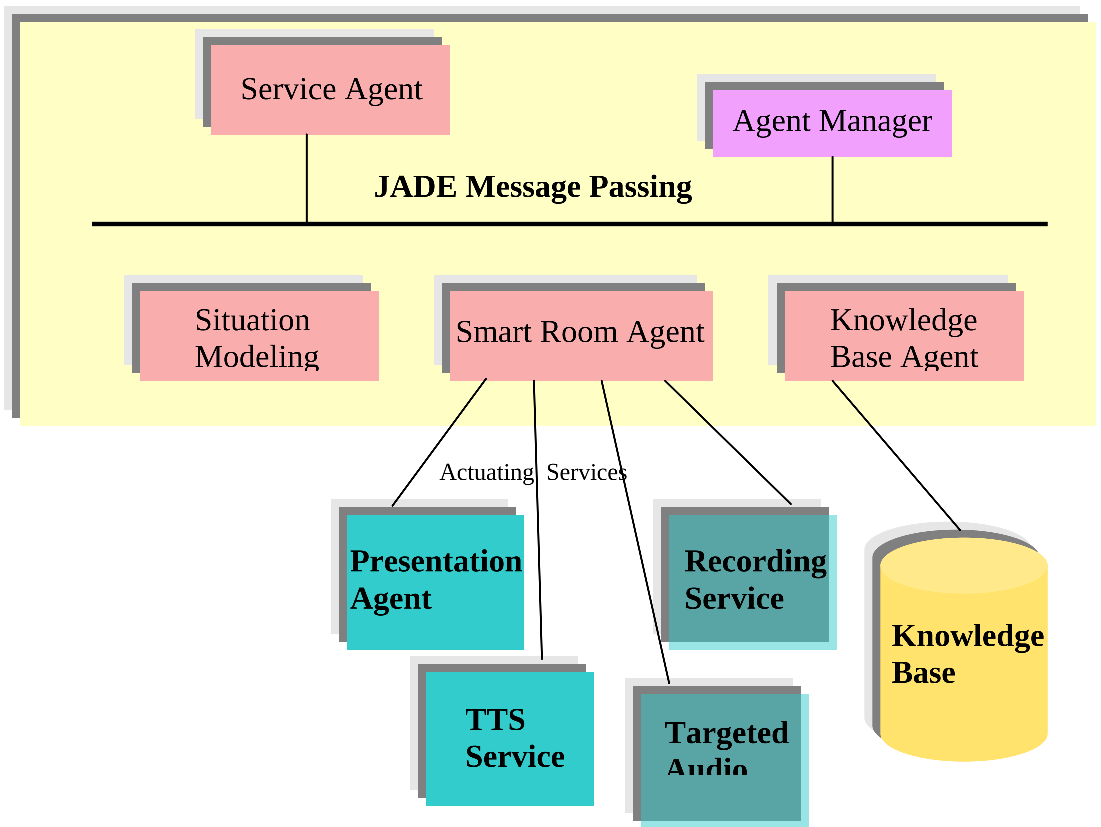


Fig. 3.5. Anatomy of the agent tier

The agent tier consists of a set of core agents:

- Device Desktop Agent, who implements the user interface required for accessing the ubiquitous services;
- Device Agent, which enables different devices to communicate with the framework;
- Personal Agent, who transmits user requests to the agent manager, which redirects them to other agents. It maintains the user's profile in order to personalize the services to the end user;
- Basic Services Agents: These agents incorporate the service logic of basic services: the ability to track composite situations, as well as the control of sensors and actuators;
- Tracking of composite situations is performed through the Situation Modeling Agent (SMA) based on the network of situations context modeling approach;
- Control of sensors and actuators is performed through the Smart Room Agent (SRA);
- The Knowledge Base Agent (KBA), allows the agents of the framework to dynamically access information on the state of the components of ubiquitous computing environment (e.g., sensors, actuators, perceptual components), through a Knowledge Base Server that is supported as an ontology management;

- Agent Manager (AM), which allows the system to be dynamically augmented with additional Service.

The framework also includes pervasive computing services, implemented as Service agents and plugged into the framework based on the JADE “pluggable” behaviours mechanism.

The communication between agents is based on standardized FIPA messages/primitives. A special agent, the Autonomic Agent Manager (AAM), deals with system failures. Its main task is to monitor specific agents, based on ping functionality (implemented as a JADE behavior).

The framework also supports dynamic registration, discovery and invocation of services, based on a proxy mechanism (using JADE), also allowing the implementation of ambient services that intelligently select the optimal actuating mechanism for a given context.

The Smart Space Resource Manager (SSRM) is an agent-based utility enabling monitoring and control of perceptual components, sensors and actuating services available. SSRM has been implemented as a JADE based multi-agent system to facilitate interoperability between the variety of hardware devices (sensing and actuating) and middleware components (perceptual and actuating) and to leverage the JADE access interface to the Knowledge Base Server, along with the corresponding Knowledge Base Agent (KBA). The developers claim that the SSRM agent is useful in the development and deployment cycle of ubiquitous computing applications, by enabling the monitoring of sensors, actuating devices and perceptual, thus helping in the identification of component failures and failing factors.

The CHIL project illustrates the fact that software agents can support the context-aware and non-intrusive nature of pervasive computing services also facilitating communications and collaboration with people, devices and other agents.

The JADE based applications case studies presented before indicate that multi-agent frameworks are able to address the various technical challenges of ubiquitous computing applications, also showing the appropriateness of using the JADE framework for the development and deployment of context-aware multi-agent systems.

IV. SYSTEM SPECIFICATIONS AND ARCHITECTURE

4.1 System functional requirements

Main system requirements are monitoring and controlling the house environment, devices, and appliances.

4.1.1 Monitoring

4.1.1.1 Context acquisition

The system will be able to gather low-level context information from a diverse range of information providers, via hardware sensors or software programs.

a) Person/objects/robots location tracking

One of the most important context attributes is location. System should detect the presence of family members or non-family members in the house, and also it should track a person’s location in a real-time basis.

b) Person/objects/robots identification

Person, object or robot identification feature will be provided on the basis of radio-frequency identification (RFID) tag technology.

c) Indoor/ Outdoor environmental conditions tracking

Environmental conditions tracking will be enabled by acquiring information from various types of sensors: lightning, pressure, humidity, noise.

d) Device, appliances tracking

As devices, appliances and actuators in the Smart House system can provide valuable context information, monitoring which of the devices is in use and their working status is one of the main tasks of the system.

The system will monitor:

- Door status (open/ closed/ locked);
- Garage gate status (open/closed);
- Window status (open/closed);
- Appliances status: cooker /coffee maker (on/off);
- Home entertainment devices status (on/off/ sound level);
- Lighting system status (on/off);
- Blinds status(on/off);
- Air conditioning status (on/off/level);

- Heating system status (on/off/level).

4.1.1.2 Context information processing

Context reasoning infers high-level contexts from sensed, atomic context data, manages context conflicts, and maintains knowledge base consistency. System will embed inference rules and also self-configuring rules, in order to support reasoning over context information.

4.1.1.2.1 Reasoning over acquired context information

High-level context information, such as current activity or user preference, will be deduced from primitive context information.

Determining the type of activity performed by an inhabitant is a complex task, and it is based on acquisition of lower level context information such as location, environment conditions tracking, or in-use device detection.

Capturing user feedback establishes the premises for acquiring personalized preference information about the house inhabitants. Preference related context information would be captured by means of special purpose interfaces for explicitly inputting user preferences and/or daily schedule.

4.1.1.2.2 Learning

Learning capabilities will be enforced in order to deduce and update user preference, based on user feedback.

4.1.1.3 Context information dissemination (sharing)

After acquiring relevant context information, context data has to be conveyed to devices and appliances. Also context information has to be adapted according to device type, device data format or whether the device is interested or not a particular type of context information.

4.1.1.4 Environment map creation with real-time updating

The system will be able to represent graphically a personalized map of the house environment, including the location of persons, objects, robots, device status, environmental conditions and deduced context information such as current activity, deduced emotional state. The system will display relevant data according to context, situation, and time.

4.1.2 Control

4.1.3.1 Context Adaptation

The Smart House system will provide automatic control of domestic systems (air condition, lights, and heating), as well as control of the basic home entertainment functions. The system will feature the capability to make a decision of what actions or missions should be taken, based on the identified context.

4.1.3.1.1 Mission management

The application will integrate predefined action plans for complex tasks, but will also provide the facility for user-defined missions. The user will have the ability to state system overall goals, such as minimizing the cost of maintaining the home or maximizing the comfort of its inhabitants.

a) Recording/ canceling a mission

The user will be able to add new missions to the system or to schedule or cancel an existing mission.

New missions can be created by the system if the current context triggers the execution of tasks that were not explicitly stated in the mission list.

b) Mission Execution

The Smart House application will automatically execute scheduled missions or may be able to reschedule failed missions.

The system commits to executing totally or in part tasks specified in the scheduled mission list, according to the associated constrains (autonomy, user and context features).

c) Mission Execution report

A mission report will be provided, including all the details of the executed mission: operations, time and mission result.

4.1.3.1.2 Operation execution

The system will be able to execute basic operations in response to context changes or primitive actions, as unit operations of a mission.

a) Objects manipulation

Robots will be able to perform object manipulations: loading and carrying objects to different locations in the house.

b) Actuator/ effectors control (turning off/on)

The system has the ability to adapt its behavior automatically, adjusting the control of devices according to current situation or direct user intervention, etc. The system will adjust the appropriate atmosphere setting by controlling the appropriate devices, according to some contextual parameters (weather conditions, internal temperature). For instance, it can set the light intensity, the room temperature (by activating the heating or the air condition), the music volume, etc.

4.1.3.1.3 User interaction

a) Confirmation messages

If the system does not have the full autonomy to execute a certain action, confirmation messages have to be generated in order to obtain the authorization of performing a sensitive action.

b) Remind/notification message

Some of the services provided to the user require generating notification messages, with the purpose of informing the user about some important issue (e.g. security issue, gas leak) or regarding the current status of an operation or mission.

c) User commands

Users may interact with the Smart House System by issuing direct commands or stating new missions on a personalized user interface.

d) Information provision

Information may be presented when explicitly requested by the user or proactively prompted by the system if it is related to the current user task.

4.2 System architecture

As I mentioned in the previous chapters, the development of a pervasive, context-aware application faces key challenges such as: user/device unawareness, interoperability, proactive behavior and heterogeneity, mobility and security. In such environments, hardware and software entities are expected to work autonomously, anticipating the behavior expected by the user.

The system architecture that I will deploy is based on the assessment of the following decisional factors:

1. System requirements analysis
2. Number of possible users (one user or many)
3. Available resources of the used devices
4. Stringent requirements for dependability, predictability, reconfigurability, awareness, accuracy, scalability, efficiency, interoperability and heterogeneity, proactivity, mobility, security and trust

5. Transparent connectivity among devices and ease of deployment
6. Further system extension

For greater flexibility and reconfigurability, the Smart House System is based on distributed, hierarchical, self-organizing, agent-based middleware architecture. The architecture comprises six collaborating layers, as shown in figure 4.1.

1. Interaction Layer

This layer handles the human-robot interaction, and the human-system interaction. It features high level capabilities for interaction between entities in the house.

This layer provides various facilities: registration of physical entities such as user or device to the system, registration of new tasks or missions (e.g. switch on/off the appliances). It also provides visual representation of the house environment to the user.

2. Context Information Layer

Context Information Layer manages context acquisition process, aggregating basic context from various sources. This layer provides the basic context model, whose goal is to represent the context based on data coming from the sensors.

The purpose of this layer is to create a context model, which provides a unique and uniform representation for context information, independently of the particular context source: sensor or device. The Interpreter Agent residing at this layer aggregates the data received from the Sensor layer according to the granularity or scope.

The Information Map Agent manages the context information provided by the context residing in the house environment: objects (doors, walls and windows), appliances, devices, inhabitants (identity, preference, and localization) but also from other context-information providers (e.g. whether prediction service). The information gathered by the Information Map Agent will be made available to the Presentation Layer.

3. Reasoning & Decision Layer

At this layer, by means of inference rules, high-level context information is inferred from basic sensed contexts and context conflicts are solved. This layer stores the **full** context model in the Context Knowledge Base while also maintaining the knowledge base consistency.

Therefore, the context Knowledge Base provides updated and persistent knowledge storage by storing the context information of every entity in the system. The Knowledge Base

Agent, operating at this layer, holds the responsibility for reasoning tasks. The outcomes of inference procedures are communicated to the Decision Agents, where they are turned into context-specific actions or missions. At this level, user preference is learned using implicit machine learning techniques.

This layer provides the following functionalities: Decision Making, Rule Expert System (performing reasoning over pre-defined rules definition and dynamic rule generation based on available contexts), reasoning about the state together with task goals and outcomes of possible actions.

The action or mission resulted from the reasoning process carried out by the Brain Agent is communicated to the Action & Mission Layer which records the action/mission, enforcing the appropriate agents to carry out the required tasks.

4. Action & Mission Layer

This layer deals with the performance of actions or missions upon the environment in order to change the current state.

Its purpose is to present system services (actions or mission) which can be provided by the system. It also has to enforce the decisions taken by the Reasoning & Decision Layer regarding the actions that should be performed on context change.

5. Sensor Layer

The Sensor Layer manages the flow of sensor information from the Physical Layer to the Context Information Layer. The Sensor Agents residing at this layer gather data regarding specific sensor information: sensor data, timestamp, sensorID, sensor accuracy (credibility). Sensor malfunctionalities and/or failures have to be detected at this layer.

6. Agent Platform Layer

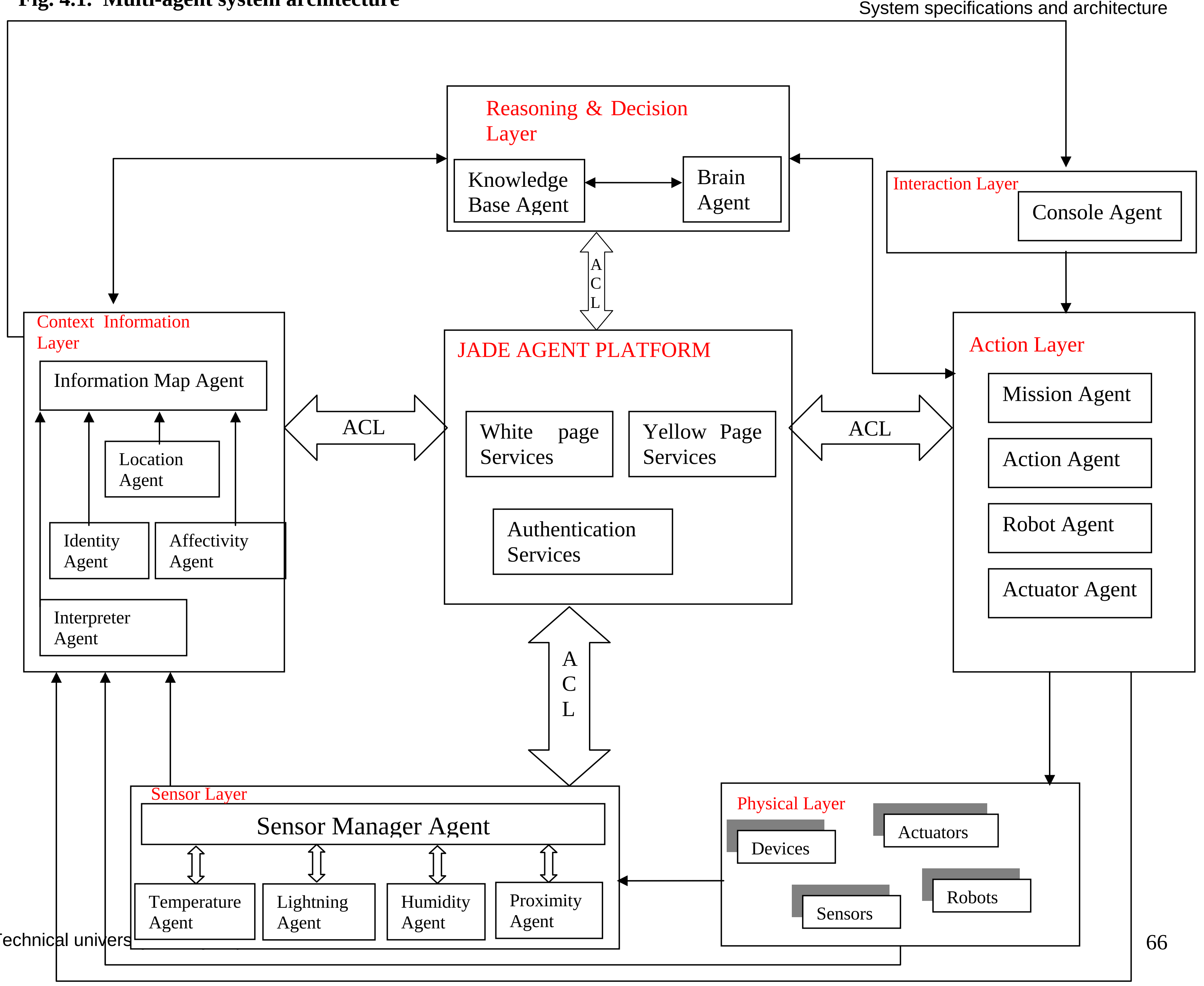
It is the deployment platform for all the agents in the Smart House System and provides transparent communication between agents deployed on different devices.

7. Physical Layer

This layer comprises physical sensors, various objects, devices, appliances and actuators (e.g. air conditioning). Devices are managed by Device Agents who gather device data and transforming the data into the appropriate format, before sending it to the Context Acquisition Layer.

Physical sensors measure the information required by the system and communicate the raw data to the Sensor layer. The physical layer performs the actions communicated by the Action Layer using actuators or device controllers (e.g. for closing the shutters, switch off the light, adjusting the sound volume).

Fig. 4.1. Multi-agent system architecture



4.3 System application design

The Smart House System application design addresses critical design decisions, presented as follows.

4.3.1 Multi-agent based system approach

The General System architecture is based on the Multi Agent System paradigm, and thus different parts of the system rely on interacting agents with specific skills and responsibilities. The Agents are developed and deployed on run-time environment based on the JADE framework. Every agent in the platform is loosely coupled and only has to focus on its main tasks.

The multi-agent systems are proved to be well suited for intelligent environments due to several key features. Multi-agent paradigm provides the means to develop scalable and robust systems, featuring decentralized control based on distributed autonomous entities, supports interaction between heterogeneous devices and flexible organization patterns (see chapter III).

4.3.2 Ontological approach for context representation

For the Context Information Layer as well as for the Reasoning & Decision Layer design decisions regarding context semantic representation have to be taken into consideration.

For context representation, an ontology-based context model will be developed. OWL (Web Ontology Language) will be used as representation language to enable expressive context description. The advantages of the using ontologies for context representation are illustrated in chapter III.

4.3.3 Rule-based reasoning to support context awareness

Reasoning & Decision Layer design decisions regarding logic-based context reasoning and knowledge sharing have to be addressed.

In order to make queries on the context information, we adopted the RDF Data Query Language (RDQL), for querying raw context information from the model. As more complex context data has to be inferred from basic atomic concepts, a rule-based approach will be adopted. The reasoner is capable of executing inference rules and handling context updates, and when provided with state information, it can interpret the information and infer new context information.

Two kinds of rules will be used: back-chaining rules (which allow us to define constraints in ontology) and forward-chaining reasoning over the OWL represented context. From the various rule-based inference engines available at the moment (Jena, OWLJessKB, Racer, FaCT, Pellet, Cerebra, DLP, etc.), on will deploy Jena rule engine to support forward-chaining reasoning over the OWL represented context.

To query the ontology models we use the Jena ARQ query engine, which supports the SPARQL RDF Query language.

V. DETAILED DESIGN

This chapter describes the Smart House detailed design procedure. In this phase, the system is designed to atomic detail, based on the Tropos Development Methodology. This section also includes a thorough presentation of the database architecture, rule expert system, user interfaces, flowcharts and communication between subsystems, development, integration and testing plans used in the development phase.

5.1. Detailed Design based on the TAOME methodology

Out of the various methodologies, which provide high level design concepts to abstract from complex system requirements and to support different design phases, I adopted the Tropos Development Methodology. The Taom4E visual modeling tool, which implements the TROPOS methodology, is used as a modeling tool for designing the Smart House multi-agent society.

Based on the Tropos iterative development process, the following design phases have been covered: Early and Late Requirements Analysis, Architectural Design, Detailed Design and the Implementation.

All the design phases are based on the common concepts of agent and agent-related notions such as goals, plans and dependencies, starting from the modeling of stakeholders and their intentions and leading to an architectural design of a multi-agent system.

5.1.1. Early Requirements Analysis

In this phase an analysis of the organizational setting is performed, by focusing on the stakeholder’s goals and dependencies with respect to the system. In an actor diagram, as shown in figure 5.1, stakeholders are modeled as social actors that depend on one another for the accomplishment of goals, plan execution and resources. For each actor, the main goals are analyzed and decomposed in a goal diagram.

The organizational setting analysis is guided by answering the following questions and graphically modeling the analysis results:

1) Which are the main actors?

The stakeholder identification in Smart House System revealed that the main actors which interact with the system are the family members (authorized to make modifications to the system and take decisions) and non-family members (guests) which have limited privileges.

2) What are their goals?

The family, as a generic actor has the following goals with respect to the system:

a) Mission Management - with the following sub-goals:
   i) Scheduling a new mission;
   ii) Canceling a scheduled mission.
b) Smart House Control - represents the ability of the Family Actor to control the house devices, sensors, actuators and robots.
c) House Map Visualization: One of the stakeholders' goals is to visualize a personalized map of the house environment, including the location of persons, objects, robots, device status, environmental conditions and deduced context information such as current activity, deduced emotional state.

3) How can they achieve them? Does an actor depend on another one to achieve its goals?

The means to achieve the state of affairs desired by the Family actor is to assign the responsibility of fulfilling the goals to the Interaction Agent, which plays the role of a broker, interfacing with the system and requesting the appropriate execution of goals and plans from the System Agents.

The next step in this phase is to analyze each goal, taking the perspective of each single actor and showing how can it be decomposed in sub-goals, alternative ways to reach a goal, means (e.g. plan /resources) to achieving them, goals which may prevent or contribute to the achievement of a goal. The results of this analysis are shown in figure 5.1.

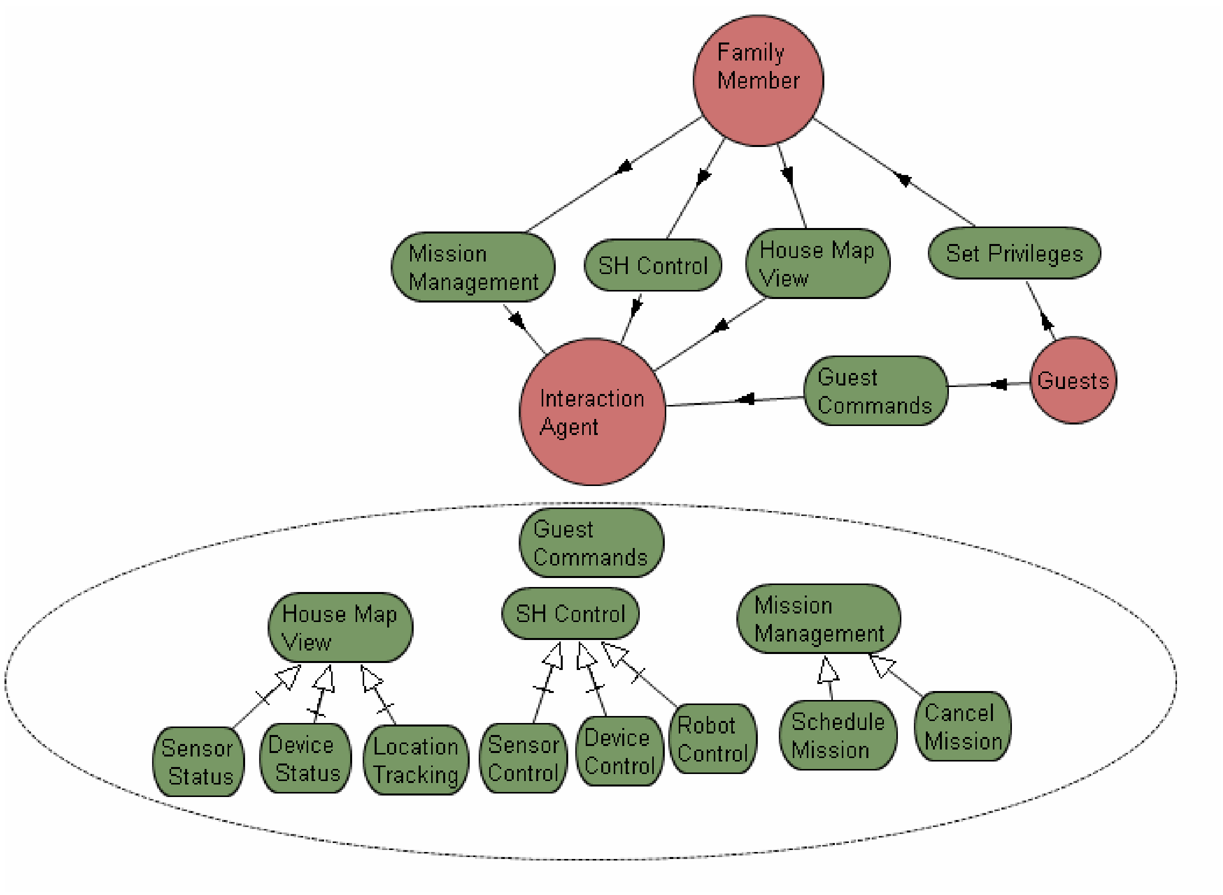


Fig. 5.1. Early Requirements Analysis

5.1.2. Late Requirements Analysis

In the Late Requirements Analysis phase, the actor diagram from the previous phase is extended by modeling the system in relation to its environment.

The stakeholders' goals are delegated to one or more system actors. These dependencies define the functional and non-functional requirements of the system. In a goal diagram, the internal system analysis is performed, starting from the delegated goals. Goals can be decomposed at a high level and contributions between goals and softgoals are examined.

The goals decomposition, means-end and contribution analysis are performed on the system's goals as captured in figure 5.2.

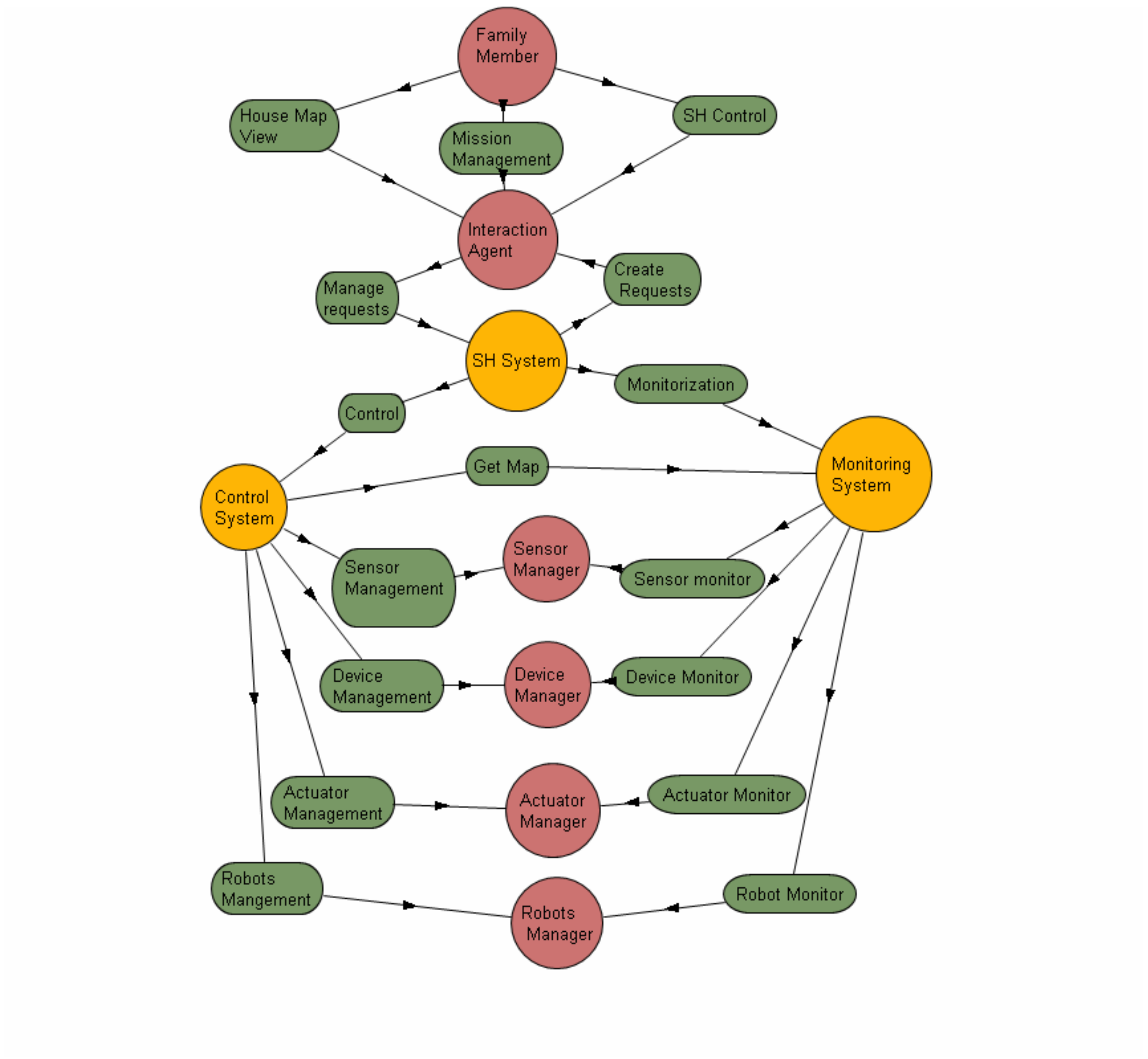


Fig. 5.2. System's goals

5.1.2.1. System actor analysis

In this phase new system actors are introduced, which are responsible with the core system goals accomplishment: Smart House System, Control System and Monitoring System.

The Smart House System actor interacts with the Interaction Agent and disseminates tasks for the Control System Actor and for the Monitoring System Actor. The Control System Actor relies on Sensor Manager, Device Manger, Actuator Manager and Robots Manager for the execution of plans and accomplishment of specific goals. The Monitoring System builds the Environment Map providing personalized information to the Family Actors by means of the Interaction Agent (Presentation Agent). The Environment Map information is processed by the Control System, which includes the decisional mechanisms for dynamically adapting to the current context information.

5.1.3. Architectural Design

In this phase the organizational architecture is developed, decomposing the system into a multi-agent system

In each architectural design diagram, roles are refined at a more detailed and precise level, by introducing new sub-actors. These agents are responsible for achieving some specific goals or performing plans emerged along the system goal analysis made in the previous phase. Additional actors are introduced to contribute to the fulfillment of some specific functional or non-functional requirements. The actors set up in this phase can be seen as software agents or, to develop a more flexible system, as roles, which can be played by an agent. Now goals and plans of the new actors are further refined on a more detailed level.

The functionalities for achieving a goal can be analyzed by defining the plans in means-end relation to the goals; then, these plans can be decomposed till down to atomic activities. The next step in this phase consists of capturing the capabilities (functionalities) that software agents or roles have to provide and that are further developed in the detailed design phase, using UML activity and sequence diagrams.

The Architectural Design phase comprises the following activities:

i) decomposing and refining the system actor diagram

This step provides the inclusion of new actors due to delegation of subgoals upon goal analysis of system's goals:

ii) inclusion of new actors according to the choice of a specific architectural style;

iii) inclusion of new actors contributing positively to the fulfillment of some non-functional requirements;

iv) identifying capabilities.

The next subchapters illustrate the architectural design diagrams resulted from the analysis performed at this phase, emphasizing the agents’ specific roles, responsibilities and functionalities.

5.1.3.1. Control System

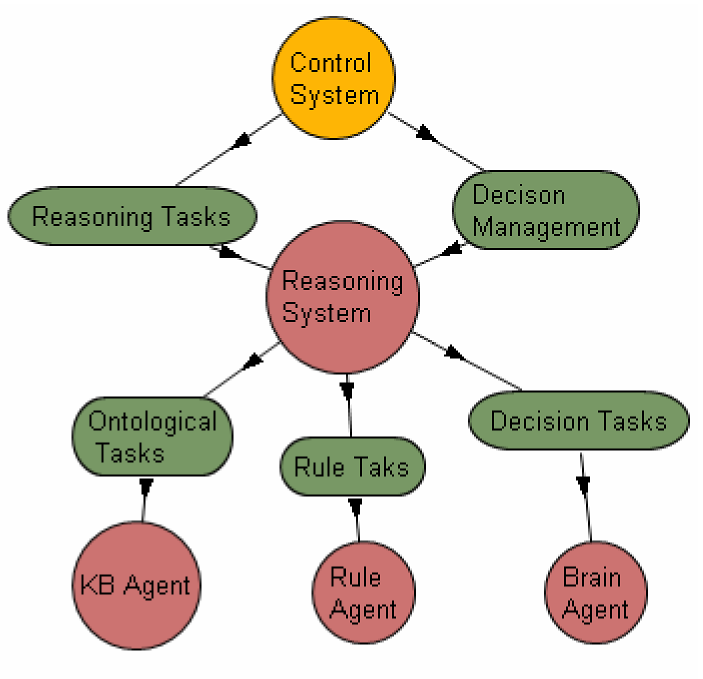


Fig. 5.3. Control System

5.1.3.2. Sensor Sub-System

The sub-actors identified for the Sensor Multi-agent Sub-System are the Reasoning System Actor, the KB Agent, the Rule Agent, and the Brain Agent. Their particular responsibilities are further detailed in the goal diagrams, highlighting the knowledge level and the agents' capabilities with the corresponding abilities (plan level).

These agents provide context acquisition functionalities; more specifically they gather low-level context information from a diverse range of sensors. The Lightning Sensor Agent, the Humidity Sensor Agent, the Temperature Sensor Agent process the low-level sensor data, using some specific methods for interpreting sensor data and communicating results to the Monitoring System Agent. The Sensor Manager has the role to create and monitor the Sensor agents, detecting failures or abnormal functionalities. It has the obligation to notice the Monitoring System Agent if an abnormal sensor behavior is detected and also the Control System, which takes the appropriate decision in this case. A detailed view of the relationships between the Sensor Layer agents is detailed in figure 5.4.

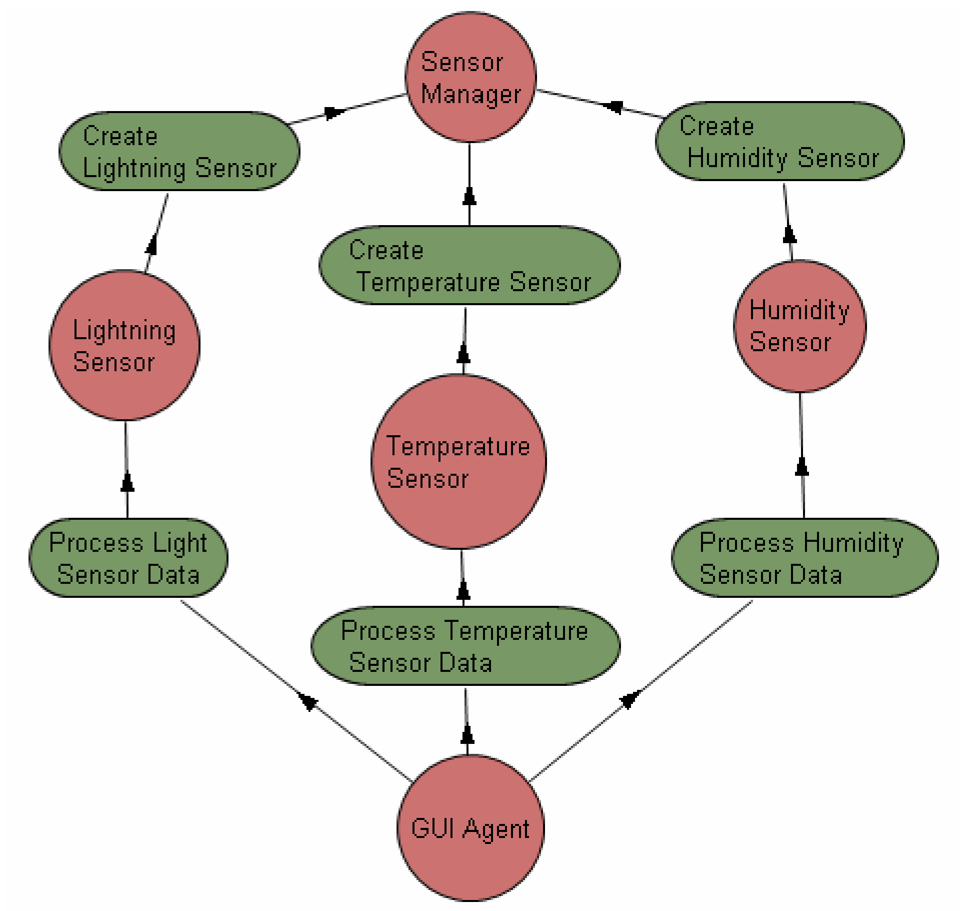


Fig. 5.4. Sensor Layer

5.1.3.3. Monitoring System

The Agents residing at the System Monitoring Agents aggregate context information related to location **tracking** (for persons, indoor objects or robots), identification, devices' and actuators' status.

The Information Map Agent aggregates the low-level context information, builds a detailed Information Map which is refined for presentation at the Interaction Layer and as input for the Control System. The Interpreter Agents deduces higher-level contexts from the atomic data provided by the Sensor System, Device System and the Robot System. It communicates the results to the Information Map Agent, which has the task to integrate the deduced information in the Information Map, as shown in figure 5.5.

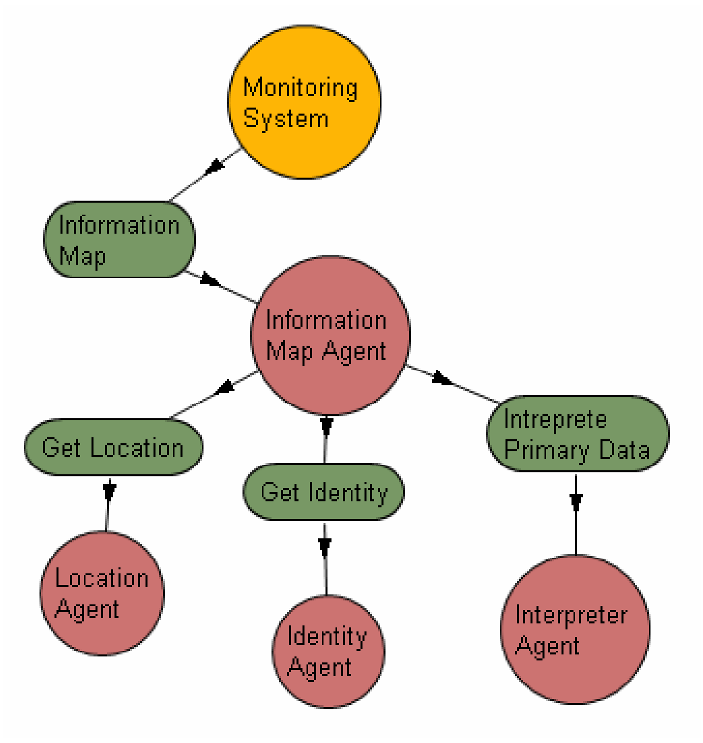


Fig. 5.5. Monitoring System

5.1.3.4. Device Sub-System

The Device Sub-System (fig. 5.6.) monitors the context-information provided by indoor equipment or devices, supervising which of the devices are in use and their working status. The following devices are monitored: home entertainment devices, air conditioning, heating system. For each device a Device Agent is delegated to interpret the signals coming from devices and to execute operations such as turning off/on or changing the device parameters.

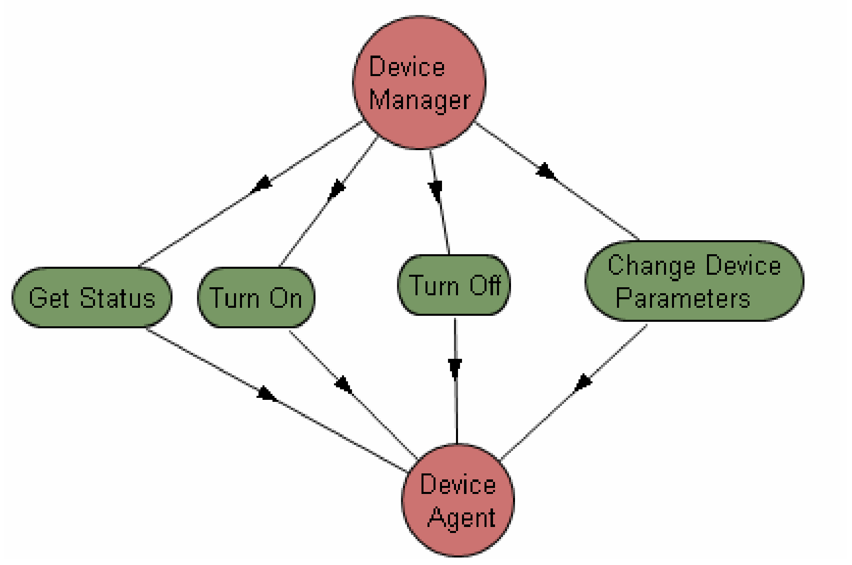


Fig. 5.6. Device Manager

5.1.3.5. Actuator Sub-System

This multi-agent sub-system manages the context-information coming from actuators, and it is responsible for issuing specific commands such as turning off/on or changing the actuator parameters. The Actuator Manager collects and interprets data provided by the Actuator Agents who are responsible for achieving goals at the level of each actuator (e.g. turn off/on actuator) (fig. 5.7.).

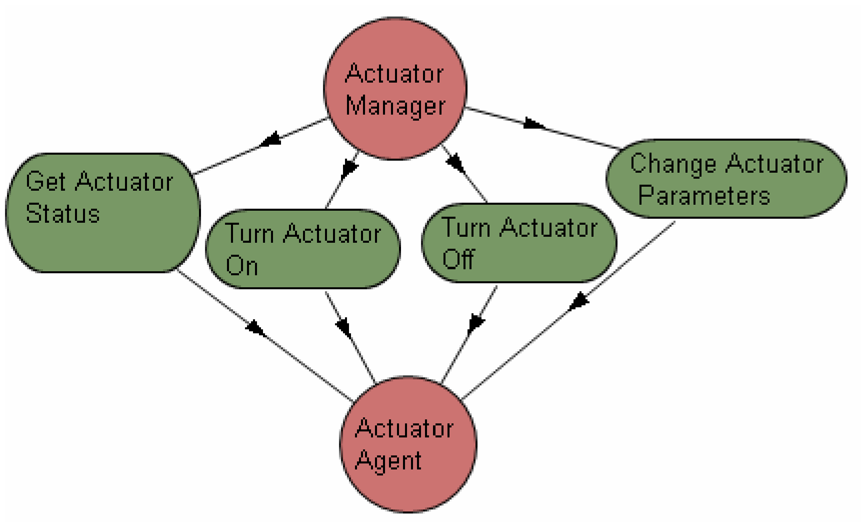


Fig. 5.7. Actuator Manager

5.1.3.6. Robot System

As the Smart House System is equipped with three robots, a multi-agent sub-system has to be design to deal with specific robot related tasks. Robots will be able to perform object manipulations: loading and carrying objects to different locations in the house. Robot Agents must also have reasoning capabilities, in order to operate autonomously, exhibiting a context-sensitive behavior. The Robot Manager Agent has the role to initiate mission for the three robot agents: the Supervisor Robot Agent, the Pet Robot Agent and the Transporter Robot Agent. Each of these agents has well defined responsibilities, and is able to execute scheduled missions or non-scheduled missions based on the current context. A detailed view of the robot sub-system diagram is presented in figure 5.8.

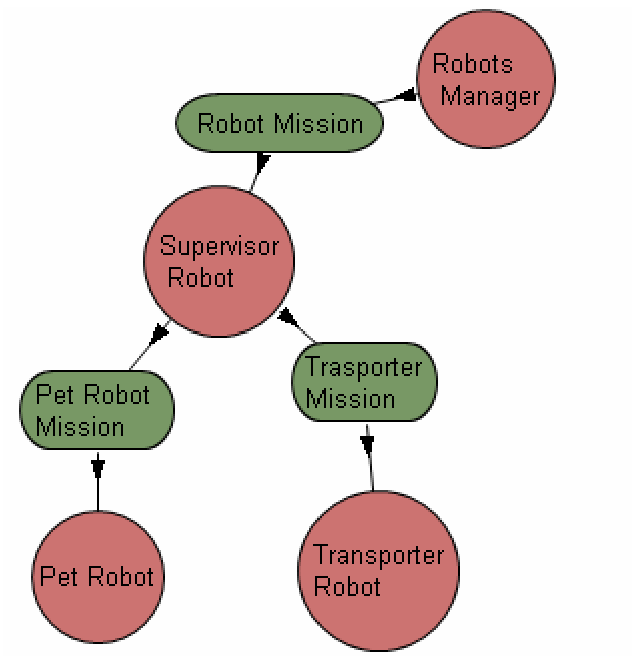


Fig. 5.8. Robots Manager

5.2. Database architecture / Rule Expert System

In order to support the reasoning and decisional mechanisms for the Brain Agent residing at the Reasoning & Decision Layer, the Smart House System must integrate two subsystems: a persistent storage for rules and context information history, as well as an Expert System. The abstract diagram of these subsystems is shown in figure 5.9.

The Rule Engine is based on the Jena2 inference subsystem, which allows the integration a range of inference engines or reasoners (Transitive reasoner, RDFS rule reasoner, OWL, OWL Mini, OWL Micro Reasoners, DAML micro reasoner, Generic rule reasoner). The Rule Engine runs the rules stored in the DB Rules and monitors the changes that occur in the values of the ontological context data. The Rule Engine supports inference on the context objects and then updates the deduced context information back into the ontology model.

The DB Rules component stores the domain rules in a given format (as required by Jena) and includes parameters to be modified by the Expert System (weight matrix values).

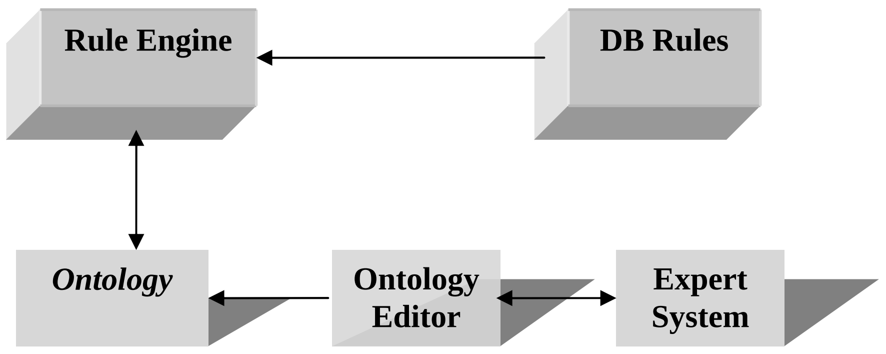


Fig. 5.9. DB and Expert System

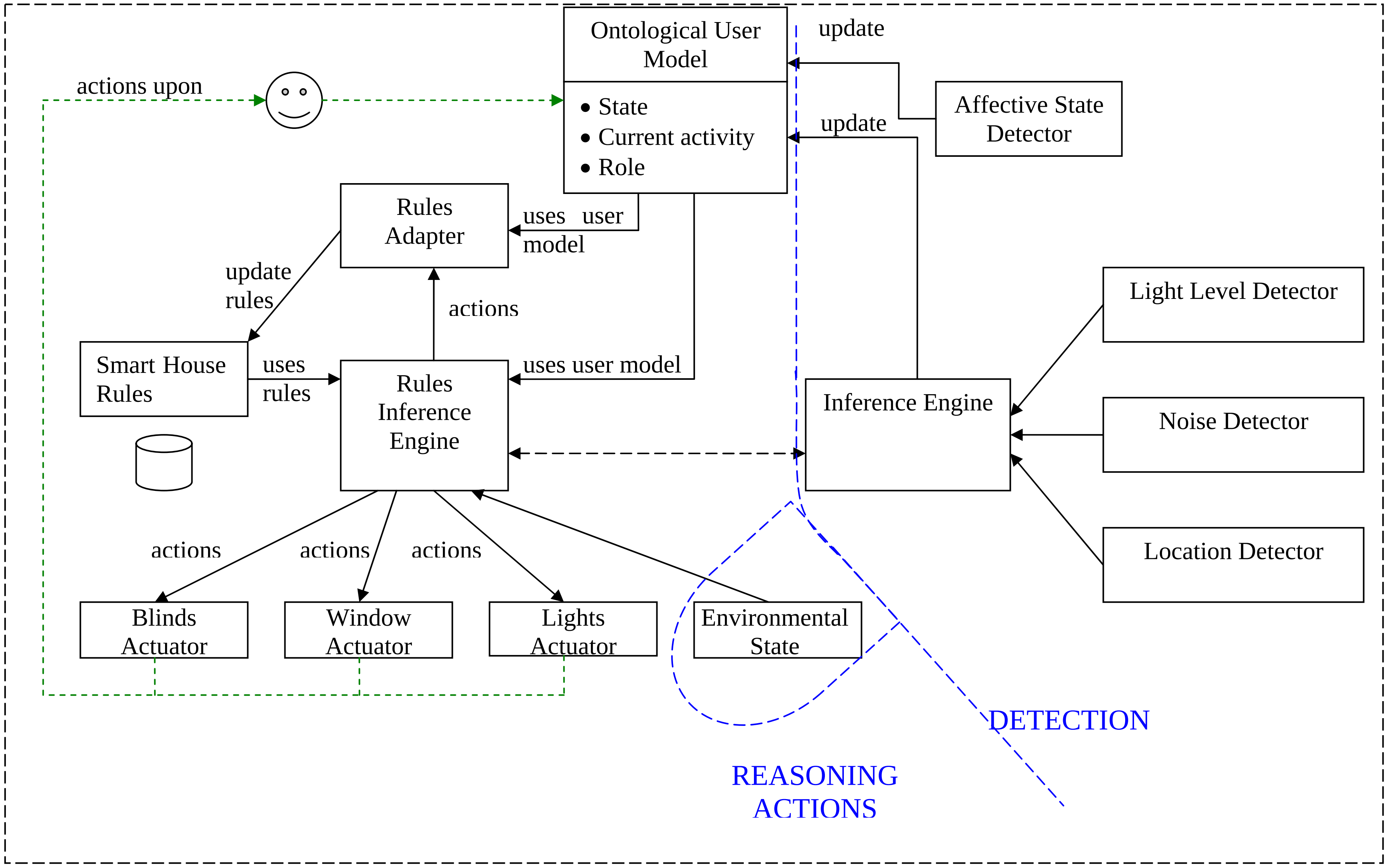
The Ontology stores the concepts, represented as classes and their relations. As the ontological model is updated with new values for the context elements (obtained from sensors, devices and other context information sources), the old values are recorded in the persistent ontology database in association with their timestamp.

The Expert System is learning, based on feedback from the user, what it would be the best response that that user would expect from the system. The neural networks will be trained by using a set of general predefined examples. After that, the system will run and adapt online based on the user’s feedback (i.e. satisfaction level measured from his affective reactions on the systems behavior).

The Ontology Editor reads the characteristics of the neural network (i.e. weight matrix, number of hidden layers, threshold function type) and stores it in the Ontology as the values change in time.

The integration of the Expert System with the other sub-components is presented in figure 5.10.

Fig. 5.10. Expert System integration



5.3. User Interface Design

The Smart House System includes several user interfaces meant to provide support to the visual representation of the context information provided by sensors, devices and objects and also visual interactivity with the System as a whole (by submitting new plans or missions, or by issuing specific commands for the indoor objects).

Two of the interfaces are designed to work on Desktop Computers.

The interfaces are designed to be invoked from Jadex agents. The interfaces gather interface data dynamically by collaborating with the appropriate agents as detailed in the system architecture diagram:

These interfaces (see Appendix) support the following functionalities:

- Visual representation of the inhabitants, robots, objects and devices;
- Visual representation of the house rooms and objects topology;
- Visual representation of deduced context information such as current activity, deduced emotional state;
- Monitoring the indoor objects' and devices' status (Doors, Windows, Appliances, Home entertainment Devices, Sensors);
- Monitoring the House Inhabitants' emotional State;
- Sending Feedback back to the system, by specifying the inhabitants' emotional state
- Mission Management.

In order to support mobility in the Smart House System, a more simplified interface is designed to be run on Mobile Devices. This interface allows the user to monitor and issue commands for the Smart House System, even when located remotely.

Due to the fact that Jadex does not fully support agent mobility, the Mobile Phone Interface Agent will be developed in Jade.

5.4. Testing Scenario

The Testing Scenario for the Smart House System is the following:

Alice and Bob Smith is a married couple who have just started using The Smart House System at their residence. They have two children: Jane (age 16) and Little John (age 4). In order to gain more effectiveness in housework, the family uses three Robots: a Supervisor Robot, a Pet Robot and a Transporter Robot.

The Transporter Robot’s main task is to carry objects from a room to another and place them on the right shelves. The Pet Robot has the role to supervise little John and to maintain a joyful emotional state for the boy. In order to achieve this goal he has several plans such as: playing some nice song, bringing cookies or candies, taking the boy to sleep if the situation is appropriate, or notifying the parents in the case it cannot achieve its goal.

The Supervisor Robot has the task to supervise the Pet Robot and the Transporter Robot. It may issue specific commands or missions for the two robots.

VI. PROTOTYPE REQUIREMENTS

The implemented prototype application is meant to run on PCs and mobile devices. For running the demo application on a PC the minimum requirements are the following:

- 1 GHz 32-bit (x86) or 64-bit (x64) processor
- 512 MB RAM
- Any OS running Java Virtual Machine

The system needs the following libraries:

- Jena Toolkit Library: http://jena.sourceforge.net/downloads.html
- Jadex http://vsis-www.informatik.unihamburg.de/projects/jadex/download.php
- Jade: http://jade.tilab.com/

The mobile agents were tested on two mobile phones, Nokia 6630 and Nokia N95. The minimum requirements for mobile phones are CLDC 1.0. and MIDP 2.0.

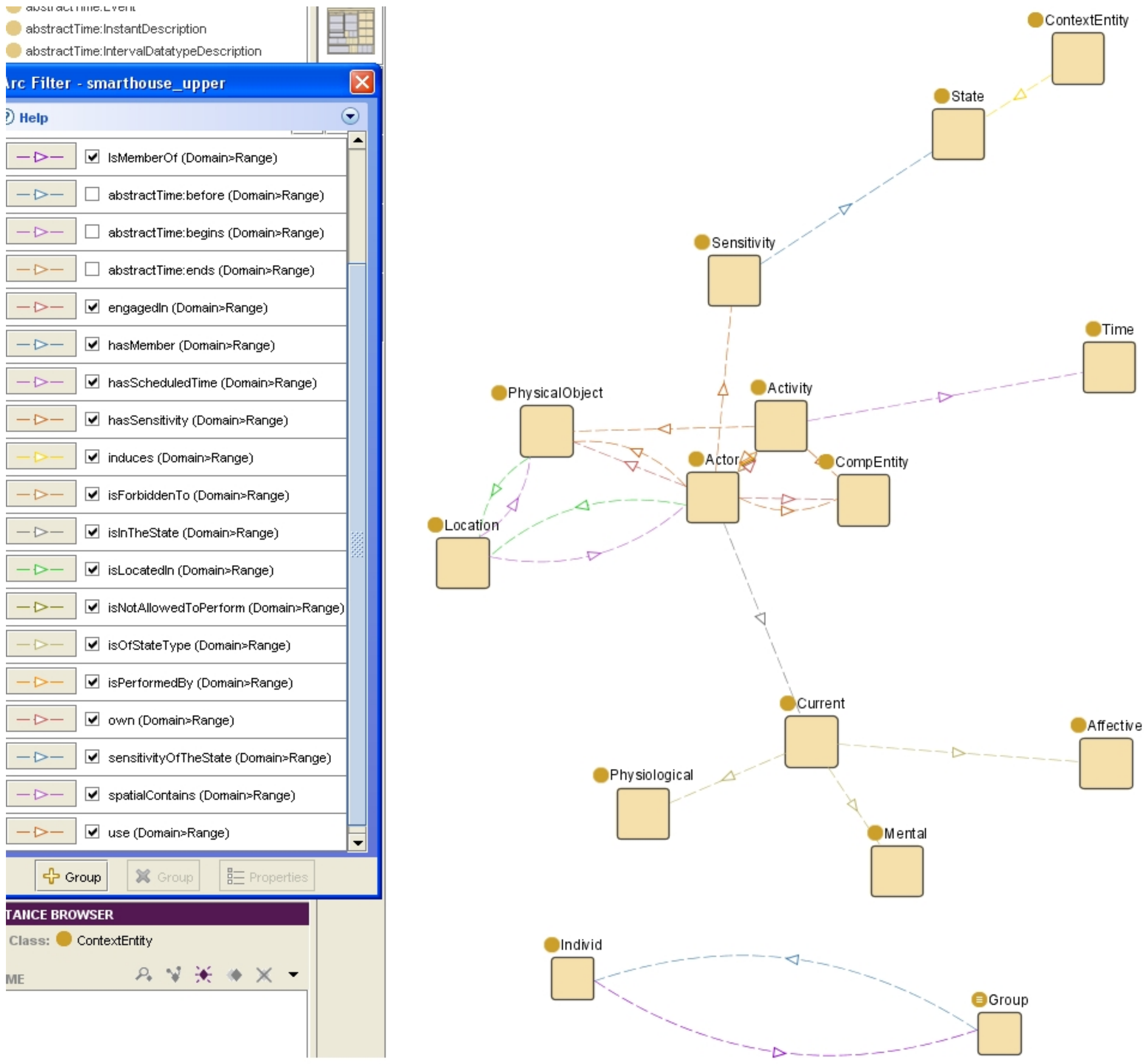
Running the demo program is done by running the file SmartHouse.bat which starts the agent platform and the necessary agents. User interaction with the system is asked on the two interfaces designed for updating and monitoring the house environment.

VII. RESULTS

7.1. Context Representation

For the representation and modeling of semantic context information, we developed an ontology-based context model using Protégé-OWL Editor and Knowledge Acquisition system.

Using Protégé OWL [62] in modeling the ontology, proved to be a good decision because along with the variety of storage formats, and data acquisition and visualization tools, it also grants high-performance classifiers. Also Protégé OWL API is written in Java and can be easily integrated into the application.


abstractTime:Event
abstractTime:InstantDescription
abstractTime:IntervalDatatypeDescription

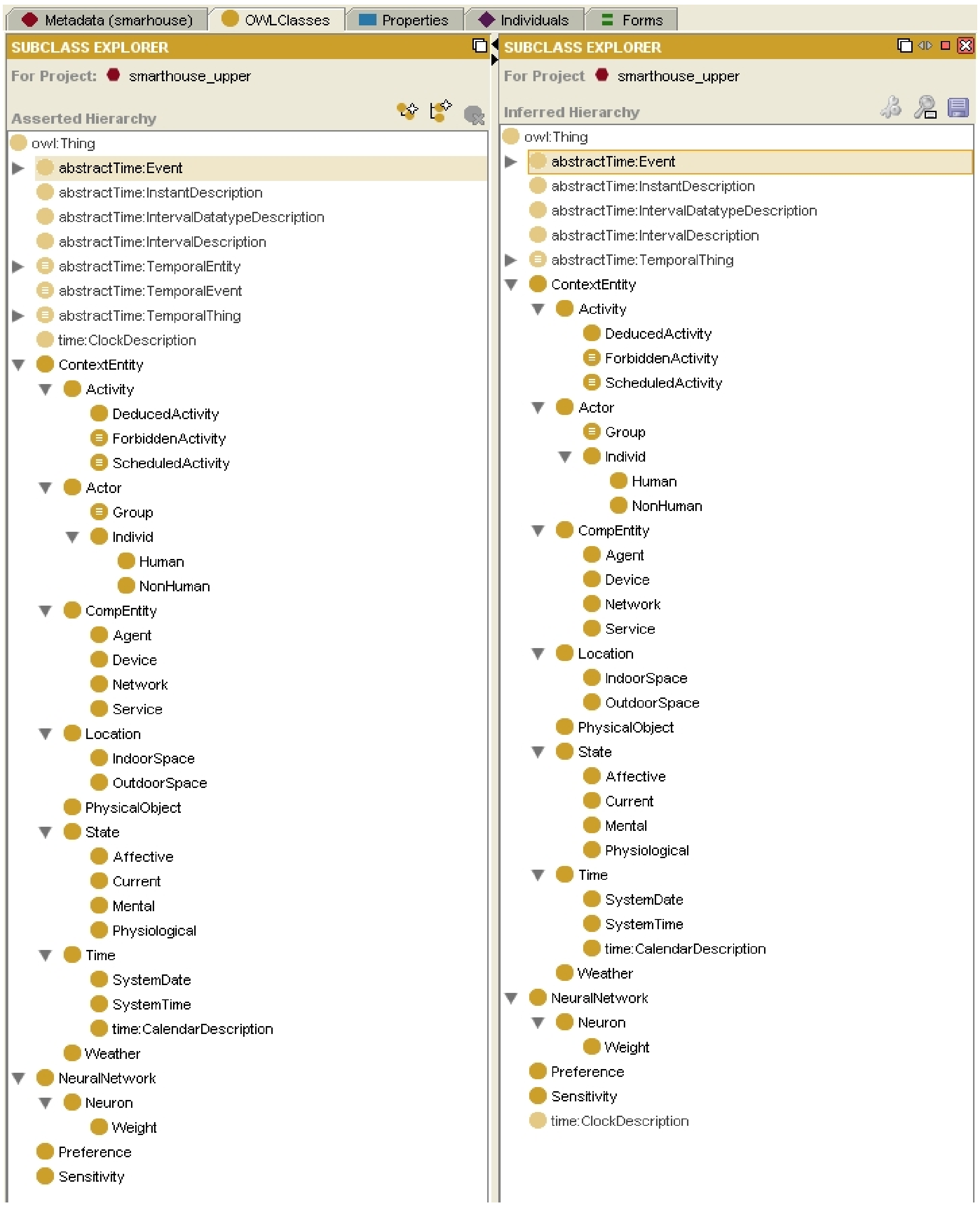
Arc Filter - smarthouse_upper

- [✓] IsMemberOf (Domain>Range)
- [ ] abstractTime:before (Domain>Range)
- [ ] abstractTime:begins (Domain>Range)
- [ ] abstractTime:ends (Domain>Range)
- [✓] engagedIn (Domain>Range)
- [✓] hasMember (Domain>Range)
- [✓] hasScheduledTime (Domain>Range)
- [✓] hasSensitivity (Domain>Range)
- [✓] induces (Domain>Range)
- [✓] isForbiddenTo (Domain>Range)
- [✓] isInTheState (Domain>Range)
- [✓] isLocatedIn (Domain>Range)
- [✓] isNotAllowedToPerform (Domain>Range)
- [✓] isOfStateType (Domain>Range)
- [✓] isPerformedBy (Domain>Range)
- [✓] own (Domain>Range)
- [✓] sensitivityOfTheState (Domain>Range)
- [✓] spatialContains (Domain>Range)
- [✓] use (Domain>Range)

INSTANCE BROWSER
Class: ContextEntity

Fig. 6.1. Visualization of the general ontology class relationships

The context model is based on a two-layer model: upper layer ontology for general concepts and a domain specific ontology, applicable to the Smart House domain. The upper layer ontology describes general, basic contextual atoms for all domains, such as *Activity, Actor, Location, Physical-Object, State, and Time*. Figure 6.1. presents a visual representation of the General Ontology class relationships, using the Jambalaya plug-in [72].



```
owl:Thing
▶ abstractTime:Event
  abstractTime:InstantDescription
  abstractTime:IntervalDatatypeDescription
  abstractTime:IntervalDescription
▶ abstractTime:TemporalEntity
  abstractTime:TemporalEvent
▶ abstractTime:TemporalThing
  time:ClockDescription
▼ ContextEntity
  ▼ Activity
      DeducedActivity
      ForbiddenActivity
      ScheduledActivity
  ▼ Actor
      Group
    ▼ Individ
        Human
        NonHuman
  ▼ CompEntity
      Agent
      Device
      Network
      Service
  ▼ Location
      IndoorSpace
      OutdoorSpace
    PhysicalObject
  ▼ State
      Affective
      Current
      Mental
      Physiological
  ▼ Time
      SystemDate
      SystemTime
      time:CalendarDescription
    Weather
▼ NeuralNetwork
  ▼ Neuron
      Weight
  Preference
  Sensitivity
```

```
owl:Thing
▶ abstractTime:Event
  abstractTime:InstantDescription
  abstractTime:IntervalDatatypeDescription
  abstractTime:IntervalDescription
▶ abstractTime:TemporalThing
▼ ContextEntity
  ▼ Activity
      DeducedActivity
      ForbiddenActivity
      ScheduledActivity
  ▼ Actor
      Group
    ▼ Individ
        Human
        NonHuman
  ▼ CompEntity
      Agent
      Device
      Network
      Service
  ▼ Location
      IndoorSpace
      OutdoorSpace
    PhysicalObject
  ▼ State
      Affective
      Current
      Mental
      Physiological
  ▼ Time
      SystemDate
      SystemTime
      time:CalendarDescription
    Weather
▼ NeuralNetwork
  ▼ Neuron
      Weight
  Preference
  Sensitivity
  time:ClockDescription
```

Fig. 6.2. Visualization of the general ontology class hierarchy

The specific ontology imports the general ontology classes and relations with specific concepts in our domain. As Protégé-OWL provides the facility to import and reuse external ontologies into an OWL ontology built in Protégé, I used this capability to integrate a more complex Time Ontology into the Specific Ontology (http://daml.umbc.edu/ontologies/cobra/0.4/time-basic#).

The root classes for the General Ontology are the ContextEntity, NeuralNetwork Preference and Sensitivity classes. As its name suggests, the ContextEntity class denotes the general types of context information (fig. 6.2.), which are relevant for the Smart House domain: *Activity, Actor, Location, Physical-Object, ComputationalEntity, State, Time, and Weather*.

From the root classes, more specific context atoms are derived (fig. 6.3.): *DeducedActivity, ScheduledActivity, Individ, Group, IndoorSpace, OutdoorSpace, Agent, Device, Network, and Service*.

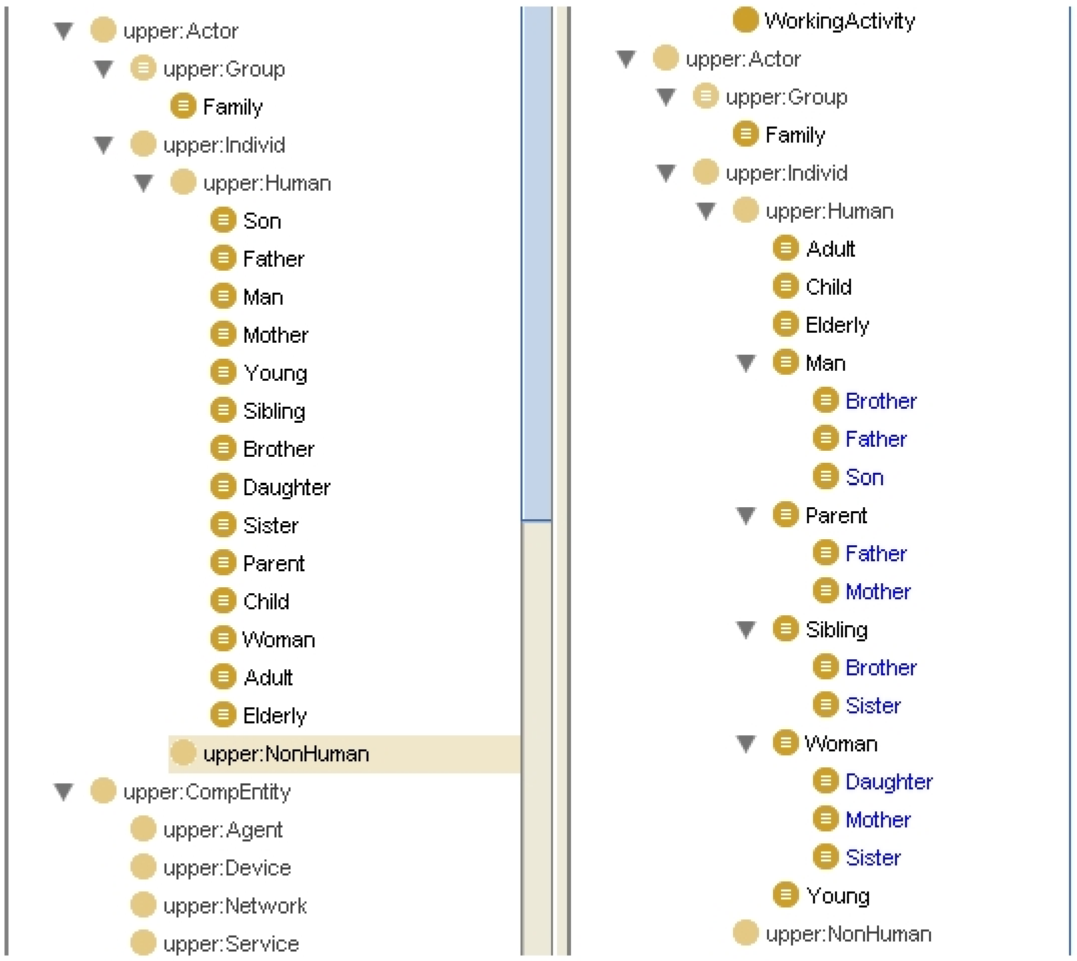


Fig. 6.3. Specific Ontology - Representation of the Asserted Ontology (left) and the Inferred Ontology (right)

Relationships between classes are modeled using Object Properties or DataType Properties. For example all the instances of class Actor are related with instances of class Activity by the ObjectProperty “has Activity”.

The Smart Home Specific ontology imports the General Ontology and adds complexity by extending the general ontology classes, adding new classes, properties and axioms. For example, in the Smart House an important aspect is the relationship between the family members in order to model user preferences and system access rights. In this case the Human class is inherited by new classes (e.g. Woman, Man, Mother, Daughter, etc.) which show the characteristics and relationships of the family members. Using ontological axioms and rules, the asserted initial ontology can be used for deducing new information, as shown in figure 6.3. The image shows the Inferred Ontology, resulted from the reasoning process over the Asserted Ontology, using Pellet Reasoner which is embedded in the Protégé framework.

After defining the class hierarchy, relations and axioms, the Smart House knowledge base was created, by defining individual instances of these classes filling in specific slot value information and additional slot restrictions.

7.2. Context Interpretation

One of the main reasons of using ontology-based context representation is to use a Reasoner to derive additional truths about the concepts we are modeling.

Generation of new context data from existing context information is enforced by two mechanisms: Ontological Reasoning and Rule Based Reasoning.

7.2.1. Ontological reasoning

Taking into account that the ontology is built using OWL DL, the subset of OWL Full which is optimized for reasoning and knowledge modeling, I tried to use its capabilities to the fullest. This implies taking advantage of the ontological reasoning services: class and instance classification, consistency checking, subsumption checking, equivalence checking, consistency checking, instantiation checking. In this manner, we can use a DL Reasoner to infer information that isn’t explicitly represented in the ontology. Inference Engine discovers unexpected subsumptions, and detects some errors.

In this context, I used the following elements, which support the reasoning process.

a) Defining characteristics for properties (Functional, Inverse Functional, Symmetric, Transitive);

For example *isLocatedIn* property is defined as a transitive property (fig. 6.4.). In this way the developer does not have to write an exhaustive set of relationships between class individuals.

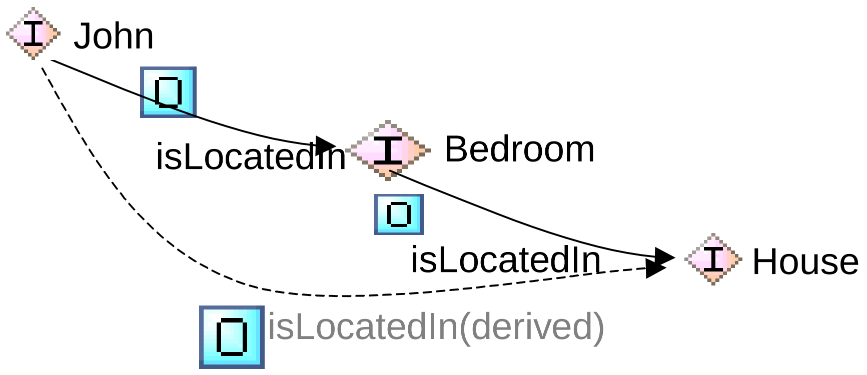


Fig. 6.4. Transitive property

b) Superclass Relationships (fig. 6.5.);

Classes are organized in a hierarchy, so that direct instances of a subclass are also instances of the hierarchical superclasses. For example if an individual is an instance of class CDPlayer, the Reasoner can infer that this individual is also an instance of the classes Device and AudioAndVideo.

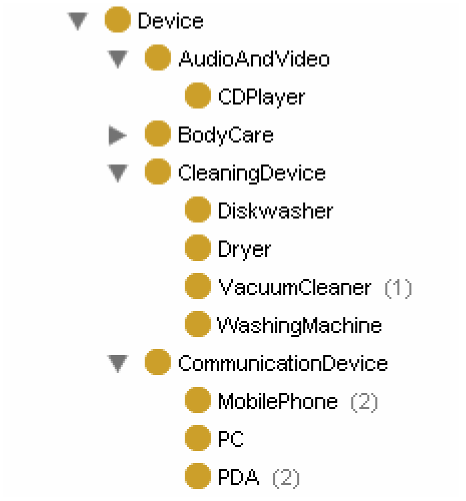


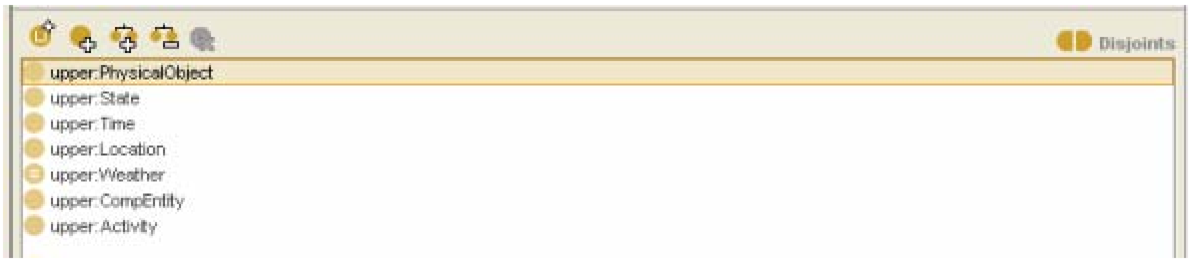
Fig. 6.5. Superclass relationships

c) Disjointness Axioms (fig. 6.6.);

OWL uses the Open World Assumption, which states that something isn't assumed to be false unless it is explicitly stated to be false.

In this case all classes could potentially overlap, unless they are explicitly stated to be disjoint with each other, or they are inferred to be disjoint with each other.

To specify that one individual can belong only to some classes, disjoint axioms are added to classes.



upper:PhysicalObject
upper:State
upper:Time
upper:Location
upper:Weather
upper:CompEntity
upper:Activity

Fig. 6.6. Disjointness Axioms

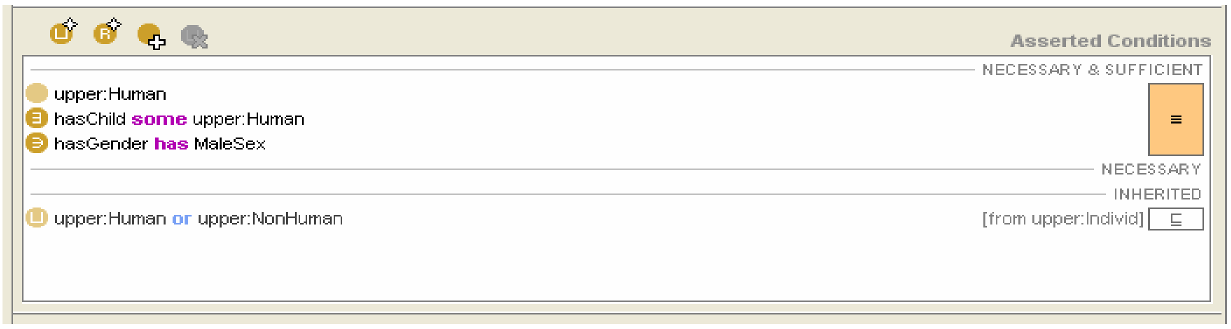
d) Restrictions;

A restriction defines an anonymous class consisting of all individuals that fulfill the condition (e.g. universal restriction, existential restriction, and cardinality restriction).

In Open World Reasoning, something isn't assumed to be false unless it is explicitly stated to be false. The general pattern for a closure axiom is to create a universal restriction along the property being closed, that has a filler which is the union of the fillers of the existential restrictions for that property.

An existential restriction describes the class of individuals that have at least one relationship along a specified property to an individual that is a member of a specified class. A universal restriction describes the class of individuals that for a given property only have relationships to individuals from a specified class.

For each class in the ontology restrictions have been defined. Figure 6.7. illustrates the conditions imposed on class Father. After performing reasoning on the ontology, class Father is correctly classified as subclass of Man and Parent. The classification results are shown in figure 6.8.



upper:Human
hasChild **some** upper:Human
hasGender **has** MaleSex

upper:Human **or** upper:NonHuman [from upper:Individ]

Fig. 6.7. Restrictions imposed on the class Father

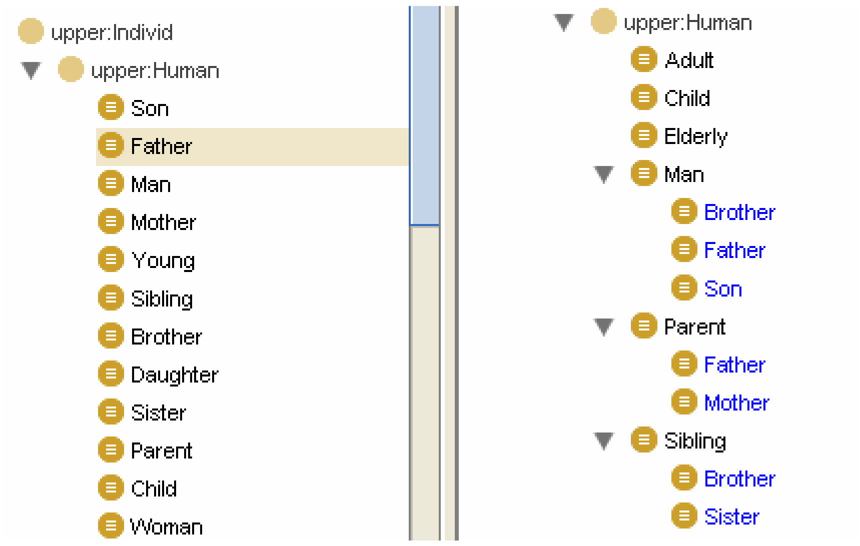


Fig. 6.8.Classification results for the class Father

7.2.2. Rules Based Reasoning

Even though Protégé plug-ins such as SWRL Tab can be used as an environment for editing and execution of SWRL rules, a general purpose API or library that can be included in the application is more desirable. Jena represents one such API for Java. As Jena supports both the RDF and the OWL API, providing limited OWL reasoning, RDF querying, in-memory and persistent storage, it was used in the project for the implementation of the Decision System (part of the Expert System).

External reasoners can be integrated to reason over Jena models, in a transparent manner. Jena is assumed to work with any conformant DIG (DL Implementation Group) reasoner such as Racer, FaCT or Pellet. We adopted Pellet, because it is robust and scalable, being the recommended Reasoner by the Jena implementers.

Jena supports a wide range of models (file-based models, database models, inference models, ontology models, memory models), and also a broad range of ontology types (RDFS, OWL, and DAML).

The first step in order to integrate our static file-based ontology designed with Protégé, was to create a Jena model. I created two Jena ontology models, a memory based model and a persistent model. The memory based model represents the current context information. When the context information is updated, the old values are stored with their timestamp in the database persistent store. The context history is kept in the persistent store; this mechanism allows applying data mining techniques over the values in the database in order to deduce new context information.

To query the ontology models I use the Jena ARQ query engine, which supports the SPARQL RDF Query language. An example of context queries, to extract sensor context information is the following:

```
PREFIX sensor: <http://www.owl-ontologies.com/SmartHouse.owl#>
SELECT ?sensorValue
WHERE {
sensor:HumiditySensor1 sensor:sensorValue ?sensorValue
};
```

A full set of context queries used in the evaluation is listed in Appendix.

Jena provides support for specifying several types of rules: deductive inference rules, constraint rules, condition/action rules, transformation rules. Jena rules are defined in the rule file. An example of deductive inference rules, to infer what is the current user activity is listed below.

```
[USER_ACTIVITY_SLEEPING:
                (?user  rdf:type pre:Human),
                (?user, pre:hasLocation, pre:Bedroom1),
                (?lightSensor, pre:hasLocation, pre:Bedroom1),
                (?lightSensor  rdf:type pre:LightSensor),
                (?lightSensor  pre:sensorValue "LOW"),
                (?noiseSensor, pre:hasLocation, pre:Bedroom1),
                (?noiseSensor  rdf:type pre:NoiseSensor),
                (?noiseSensor  pre:sensorValue "LOW"),
-> (?user  pre:hasActivity pre:Sleeping1), print(?user)]
```

Based on this rule, if one of the persons in the house is located in the bedroom, the light sensors and noise sensors indicating a “LOW” value, the deduced activity is “sleeping”.

7.2.3. Ontology as a communication vocabulary for software agents

One important property in MAS is the autonomy of agent knowledge: agents do not have direct access to the data inside other agents. Therefore we have to use messages to exchange data among agents. Message based communication between agents requires encoding the message content before sending it and decoding the message at destination. In the context of multi-agent systems, the so-called content languages and ontologies are responsible for describing how data should be encoded into messages.

The content language and the ontology specification ensure a common understanding among agents. A content language defines the syntactical mechanism used to represent data and the ontology specifies the meaning of the concepts used in the message. Smart House multi agent system uses the Nuggets XML content language. Nuggets XML is a Jadex-proprietary content language that uses the bean encoder available in the JDK, in order to convert Java objects adhering to the JavaBeans specification to standardized XML files. These XML files are then sent to the receiver agent, which decodes them by realizing the inverse procedure.

The ontology was created to achieve 2 goals: to provide a structured model of the domain knowledge and to ease the creation of JavaBean objects. The ontology was created using the Protege software with the Beanynizer plugin provided by Jadex. This plugin allows exporting all the concepts and actions included in the ontology as JavaBean Objects compatible with Nuggets XML. The use of these two tools makes the creation and modification of the domain knowledge really straightforward.

7.3 Multi Agent System implementation details

This section underlines some implementation decisions that I consider significant mentioning and that illustrate the features offered by Jadex.

7.3.1. Jadex Meta-Level Reasoning

Meta-level reasoning in Jadex can be achieved through the use of metagoals and meta-plans. Whenever an event happens (such a received message) or a goal is activated, and that situation triggers multiple matching plans, meta-level reasoning can be used to choose which plans to execute. If the event or goal has been associated with a meta-goal, this meta-goal is also activated and triggers a meta-plan. When the meta-plan finishes (i.e. the meta-goal is achieved), its result contains the selected plans, which are afterwards scheduled for execution.

In Smart House system, I decided to use meta-level reasoning to determine the agent’s strategies at runtime. In order to decide the proper action, I have implemented different plans corresponding to the agents’ different strategies.

7.3.2. Jadex Query Language

Jadex provides an OQL-like (Object Query Language) query syntax, which can be used in conjunction with any other expression statements. OQL is an extension of SQL for object-oriented databases, which is the case of the belief base of a Jadex agent.

In Smart House system, I used OQL the context-map agents to perform queries on the set of current context-information (e.g. queries for retrieving the registered sensors, devices, sensor values or device status).

VIII. CONCLUSIONS

This chapter will summarize the important insights and conclusions of this thesis.

The desiderata summarized in Chapter I have been met by the design and proof of concept implementation that validates the design of the Smart House context aware multi-agent system.

In the theoretical part of this thesis I focused on presenting the state of the art in the domain of context-aware, ubiquitous systems. I identified the issues and challenges faced by context-aware applications, and the existing theoretical or practical approaches in this field.

I have motivated the need to provide a flexible and extensible method of specifying the context model, an efficient context processing mechanism which would enable both context sharing and context reasoning. I illustrated the advantages of designing an ontology-based context model considering the fact that they allow performing a formal analysis of the domain knowledge. From the point of view of reasoning, ontologies support the derivation of implicit concept hierarchies, deducing high-level context from low-level context information, consistency checking and validation.

Additionally, I have focused on a comparative-analysis of these context-aware systems which use ontologies for context representation (CoOL [26], SOUPA [25], SOCAM [13]), and on analyzing the manner in which the ontologies were designed and integrated into dynamic pervasive environments. I also tried to evaluate the methods of providing probabilistic extensions to ontologies to support uncertainty. I concluded that although these approaches appear to solve the uncertainty issue, the main shortcomings of these solutions is that their support for uncertainty is application-specific; the models lack reusability and provide no systematic method to support uncertain reasoning mechanism.

Even if it is out of the scope of this thesis to provide a new solution of design for probabilistic ontology-based models, this analysis helped me to identify the problem that I am actually trying to solve. The general conceptual problem that I am dealing with is designing architecture to enable online adaptation to context changes in pervasive environments. To tackle this problem I led an intensive study of the advantages of using the multi-agent technology for creating flexible systems which can easily adapt to environmental changes, and are able to integrate heterogeneous components, by providing the flexibility to manage large scale, complex, distributed systems. The agent technology provides many different solutions to commonly known problems faced in the fields of distributed intelligence and ubiquitous computing. I concluded that the intelligent software agents represent the response to the challenge of high complexity in ubiquitous computing domain. Applications using this paradigm can benefit of the migration facility of the agents, and of the agents' ability to explore and discover context information, resources, services, and to communicate and cooperate with other software agents.

I have chosen to model the Smart House multi-agent architecture using the BDI-agent paradigm that allows for the development of so-called intelligent agents – agents that can reason and act based on their beliefs and intentions.

The Smart House system architecture that I proposed consists of decentralized software components, agents, which can communicate with each other. The architecture is based on distributed, hierarchical, self-organizing, agent-based middleware architecture. The architecture comprises six collaborating layers: Interaction Layer, Context Information Layer, Reasoning & Decision Layer, Action & Mission Layer, Sensor Layer, and Physical Layer. Agents are responsible for achieving specified goals, which they process independently or work in collaboration with other agents.

The Smart House system collects information from sensors, devices and actuators, recognizes the context based on this information, maintains context map, adapts to context changes by using the reasoning facilities embedded in the Expert System, and performs the chosen action plans. The system performs these activities constantly in the background, only requiring attention from the user when a confirmation is needed when action results in information to be presented to the user.

The decision making system design is based on three components: The Ontology, the Rule Engine and the Expert System. The purpose of the decision making system is to enforce an adaptive behavior by taking into account and learning from the users' feedback to system's actions. The Expert System uses a Neural Network that is trained with a predefined rules set, in order to respond as expected by the user. The important feature of this system is the ability to learn, at run-time and to adjust its predefined parameters based on the inhabitants' emotional reactions with respect to the systems decisions. Although we have defined a general architecture for the Expert System, the actual implementation and integration in this project is out of the scope of the thesis.

This thesis dissertation has presented the requirements for architecture for supporting the designing, building, execution and evolution of context-aware applications. The architecture supports a simple design process for developing the Smart House application, using the Tropos agent development methodology, centered on the design of intelligent BDI agents.

The proposed architecture also allows the exploration of challenging research areas within the field of context-aware, ubiquitous applications: starting from multi-agent system technology in conjunction with developing an ontology-based context model and the use of AI techniques for learning and decision-making process.

IX. FUTURE DEVELOPMENTS

In the this section, I will provide some directions for future developments in the Smart House System and for future research in the field of context-aware applications.

As I stressed earlier, the implementation of a complete application, integrating all the requirements of a Smart House, is a matter of future developments. For an operational Smart House application, the system should integrate a number of additional components: Sensor Network, Emotional State Detector, a complete and functional version of the Expert-System.

Another aspect which should be addressed by future work is the fact that the Jadex technology does not provide full agent migration capabilities. This limitation can be overcome at the moment by developing Jade agents for mobile devices. Moreover, further investigations and studies are needed to explore and utilize the features of the proposed multi-agent and Expert System architecture, and deploying context reasoning in a real environment.

From the point of view of the challenges that still remain to be addressed in the context of context-aware applications and ambient intelligence, we can name the following:

- Security issues

Mobile agents add more complex problems to security, as they are able to migrate and execute their code on remote platforms. Future agent implementations must also concentrate on issues related to software agent verification, restricting agents' right to do different operations on an agent platform or to make use of sensitive resources on the host on which they reside.

- Data privacy issues

Especially the privacy aspect of agent technology is an intensive research area in which the context privacy issues are not completely solved.

- Uncertainty management

Uncertainty management is important issue especially when developing applications for highly complex, dynamic environments. More extensive studies are needed in this field to provide an operational probabilistic model for managing uncertainty.

- Interoperability of context ontology models

For applications with complex and large context information models, interoperability with existing ontology is necessary. As different ontologies may model the same context information differently, a method ensures to context interoperability between different ontology, is required.

Beside the research challenges already mentioned, lots of research issues are still of high concern such as:

- context sensing and acquisition;
- context information discovery and sharing;
- context modeling and management;
- processing, aggregation and reasoning on contextual data;
- knowledge discovery and mining over historical context data;
- service-oriented architecture of context-aware systems;
- user interfaces for context-aware applications.

X. BIBLIOGRAPHY

[1] Ronny Haryanto, Context Awareness in Smart Homes to Support Independent Living, Master of Science in Internetworking, University of Technology, Sydney, 2005.

[2] Dey, Anind K. & Gregory D. Abowd, Towards a Better Understanding of Context and ContextAwareness", GVU Technical Report GIT-GVU-00-18, GIT, 1999.

[3] Held Albert, Sven Buchholz & Alexander Schill, Modeling of Context Information for Pervasive Computing Applications", Sixth World Multiconference on Systemics, Cybernetics and Informatics, SCI2002, Orlando, July 2002.

[4] Harter Andy, Andy Hopper, Pete Steggles, Andy Ward, Paul Webster, The Anatomy of a Context-Aware Application, Wireless Networks 1 (2001) 116, 2001.

[5] O. Bucur, P. Beaune, O. Boissier, Steps towards making contextualized decisions: How to Do What You Can, with What You Have, Where You Are, Lecture Notes in Computer Science, vol. 3946, 2006, p. 62-85, Septembre 2005.

[6] Luca Buriano, Marco Marchetti, Francesca Carmagnola, Federica Cena, Cristina Gena, Ilaria Torre, The Role of Ontologies in Context-Aware Recommender Systems, In The Proceedings of MOSO, 2006.

[7] Ghita Kouadri Most´efaoui, Jacques Pasquier-Rocha, Patrick Br´ezillon Context-Aware Computing: A Guide for the Pervasive Computing Community, Pervasive Services, ICPS 2004 IEEE/ACS International Conference, July 2004.

[8] Kavi Kumar Khedo, Context-Aware Systems for Mobile and Ubiquitous Networks, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006.

[9] Thomas Strang Claudia LinnhoffPopien, A Context Modeling Survey, Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp, 2004.

[10] Matthias Baldauf, Schahram Dustdar, Florian Rosenberg, A survey on context-aware systems, International Journal of Ad Hoc and Ubiquitous Computing, Volume 2, Issue 4, June 2007

[11] O. Gassmann, H. Meixner, Sensors in Intelligent Buildings, Wiley-VCH Verlag GmbH, 2001.

[12] Leijdekkers, Peter & Valerie Gay, Personalized Service and Network Adaptation for Smart Devices." IEEE Asia Pacific Conference on Communications, Perth, Australia, October 2005.

[13] Tao Gua, Hung Keng Punga, Da Qing Zhangb, A service-oriented middleware for building context-aware services, Journal of Network and Computer Applications archive Volume 28 , Issue 1, January 2005.

[14] Chen, Guanling, David Kotz, A Survey of ContextAware Mobile Computing Research, Dartmouth Computer Science, Technical Report, 2000.

[15] Chen, H., An Intelligent Broker Architecture for Pervasive Context-Aware Systems, PhD Thesis, University of Maryland, Baltimore, 2004.

[16] Dey, Anind K., Raffay Hamid, Chris Beckmann, Ian Li, Daniel Hsu, A CAPpella: Programming by Demonstration of ContextAware Applications, Proceedings of the SIGCHI conference on Human factors in computing systems, 2004.

[17] Bardram, Jakob E. The Java Context Awareness Framework (JCAF) A Service Infrastructure and Programming Framework for ContextAware Applications,Third International Conference on Pervasive Computing, vol. 3468 of Lecture Notes in Computer Science, Munich, Germany, May 2005.

[18] Mayrhofer Rene, An Architecture for Context Prediction, PhD Thesis,University Linz, Austria, 2004.

[19] Gu T., Wang X. H., Pung H. K., Zhang D. Q., Ontology Based Context Modeling and Reasoning using OWL, San Diego, USA, January 2004.

[20] Chen H., Finin T., Joshi, A. Using OWL in a Pervasive Computing Broker. In Proceedings of Workshop on Ontologies in Open Agent Systems (AAMAS 2003), 2003.

[21] Strang, T. and Linnhoff-Popien, C., A Context Modeling Survey, First International Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp, 2004.

[22] Salber, Daniel, Anind K. Dey & Gregory D. Abowd. 1999, The Context Toolkit: Aiding then Development of ContextEnabled Applications, Pittsburgh, USA, May 1999.

[23] Fahy, Clarke, CASS – a middleware for mobile context-aware applications, Workshop on Context Awareness, MobiSys, 2004.

[24] Gaëtan Rey, Joëlle Coutaz, James L. Crowley, The Contextor: a computational model for contextual information, WorkShop UBICOMP, 2002.

[25] Harry Chen, Filip Perich, Tim Finin, Anupam Joshi, SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications, Mobile and Ubiquitous Systems: Networking and Services, MOBIQUITOUS 2004, 2004.

[26] Thomas Strang1, Claudia Linnho-Popien, Korbinian Frank, CoOL: A Context Ontology Language to enable Contextual Interoperability, In Distributed Applications and Interoperable Systems (DAIS2003), 2003.

[27] Zhongli Ding and Yun Peng, A Probabilistic Extension to Ontology Language OWL, Proceedings of the 37th Hawaii International Conference On System Sciences (HICSS-37), January 2004.

[28] Tao Gu, Hung Keng Pung, Da Quing Zhang, A Bayesian approach for dealing with application specific contexts, Proceedings of the Second International Conference on Pervasive computing, Vienna, Austria, April 2004.

[29] Binh An Truong, Young Koo Lee, Sung Young Lee, A Unified Context Model: Bringing Probabilistic Models to Context Ontology, EUC Workshops 2005, 2005.

[30] Reto Krummenacher, Thomas Strang, Ontology-Based Context Modeling, In Third Workshop on Context Awareness for Proactive Systems (CAPS'07), 2007.

[31] Md. Kamrul Hasan, Kim Anh, Lenin Mehedy, Young-Koo Lee, Sungyoung Lee, Conflict Resolution and Preference Learning in Ubiquitous Environment, Lecture Notes in Computer Science, Springer Berlin, Volume 4114/2006, 2006.

[32] Reto Krummenacher, Holger Lausen, Thomas Strang, Jacek Kopeck´y Analyzing the Modeling of Context with Ontologies, In Int'l Workshop on Context-Awareness for Self-Managing Systems, May 2007.

[33] Rem Collier, Gregory O'Hare,Terry Lowen, Colm Rooney, Beyond Prototyping in the Factory of Agents, 3rd Central and Eastern European Conference on Multiagent Systems (CEEMAS'03), Lecture Notes in Computer Science (LNCS), 2691, 2003.

[34] N. Hristova, G.M.P. O'Hare & T. Lowen, Agent-based Ubiquitous Systems: 9 Lessons Learnt, In Proceedings of the System Support for Ubiquitous Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing (UbiComp'2003) Seattle, Washington, 2003

[35] B. De Carolis, G. Cozzolongo, S. Pizzutilo, and V.L. Plantamura, Agent-Based Home Simulation and Control, ISMIS 2005, Saratoga Springs, NY, USA, 2005.

[36] Oana Bucur, Philippe Beaune, Olivier Boissier, Representing Context in an Agent Architecture for Context-Based Decision Making, proceedings of CRR'05 Workshop on Context Representation and Reasoning., 2005

[37] Pavlos Moraitis, Nikolaos Spanoudakis, The Gaia2jade process for multi-agent systems development, Applied Artificial Intelligence 20(2-4): 251-273, 2006

[38] J.K. Soldatos, Software Agents in Ubiquitous Computing: Benefits and the CHIL Case Study, invited paper in the Proceedings of the Software Agents in Information Systems and Industrial Applications (SAISIA 06') workshop, Karlsruhe, Germany, February 2006.

[39] Katia P. Sycara, Multiagent Systems,. AI Magazine 19(2): 79-92 (1998).

[40] Nguyen G., Dang T.T, Hluchy L., Laclavik M., Balogh Z., Budinska I. Agent platform evaluation and comparison, Jun 2002.

[41] Anders Kofod-Petersen A Case-Based Approach to Realising Ambient Intelligence among Agents, Thesis for the degree doctor scientiarum Trondheim, May 2007.

[42] Russell, Norvig Artificial Intelligence: a Modern Approach, Russell and Norvig 1995.

[43] Fabio Bellifemine JADE Tutorial for beginners, mia.ece.uic.edu/~papers/MediaBot/JADEProgramming-tutorial-for-beginners.pdf

[44] Hristova, N., O'Hare, G.M.P., Ad-me: A Context-Sensitive Advertising System, In Proc. of 3rd International Conference on Information Integration and Web-based Applications and Services (IIWAS), September 2001.

[45] O'Hare, O'Grady, Gulliver's Genie: A Multi-Agent System for Ubiquitous and Intelligent Content Delivery, In Press, Computer Communications, Vol. 26, Issue 11, Elsevier Press. (2003) 1177-1187, 2003.

[46] Keegan, S. & O'Hare, G.: EasiShop: Context sensitive Shopping for the Mobile User through Mobile Agent Technology, Proc. of 13th PIMRC, IEEE Press, Portugal, 2002.

[47]Smith, D. C., A. Cypher and J. Spohrer (1994), KidSim: Programming Agents Without a Programming Language, Communications of the ACM, 37, 7, 55-67

[48]Hayes-Roth, B. (1995), An Architecture for Adaptive Intelligent Systems, Artificial Intelligence:Special Issue on Agents and Interactivity, 72, 329-365.

[49]Wooldridge, Michael and Nicholas R. Jennings (1995), "Agent Theories, Architectures, and Languages:a Survey" in Wooldridge and Jennings Eds., Intelligent Agents, Berlin: Springer-Verlag, 1-22.

[50] Brustoloni, Jose C., Autonomous Agents: Characterization and Requirements, Carnegie Mellon Technical Report CMU-CS-91-204, 1991, Pittsburgh: Carnegie Mellon University

[51] M.Coen, http://www.ai.mit.edu/people/sodabot/slideshow/total/P001.html

[52] JESS: http://www.jessrules.com/

[53] JADE Tutorial for beginners, organized by the JADE Board, http://jade.tilab.com

[54]Rafal Leszczyna, Evaluation of Agent Platforms, 2004

[55] Anders Liljedahl, Evaluation of Multi-Agent Platforms for Ubiquitous Computing, 2004

[56] B. De Carolis, G. Cozzolongo, S. Pizzutilo, and V.L. Plantamura , Agent-Based Home Simulation and Control

[57] J.K. Soldatos, Software Agents in Ubiquitous Computing: Benefits and the CHIL Case Study

[58] Fausto Giunchiglia ,John Mylopoulos, Anna Perini,The Tropos Software Development Methodology: Processes, Models and Diagrams

[59]http://ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html

[60] [Rao and Georgeff, 1991] A. S. Rao and M. P. Georgeff. “Modeling rational agent within a BDI architecture”. In Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR’91), Cambridge, MA, 1991.

[61] Sun Microsystems. Java Beans, http://java.sun.com/products/javabeans/

[62] Protege http://protege.stanford.edu/

[63] JADE (Java Agent DEvelopment Framework), http://jade.tilab.com/

[64] Alexander Pokahr, Lars Braubach, Winfried Lamersdorf. Jadex: A BDI Reasoning Engine, Chapter of Multi-Agent Programming, Kluwer Book, Editors: R. Bordini, M. Dastani, J. Dix and A. Seghrouchni. http://www.cs.uu.nl/docs/vakken/map/jadex.pdf

[65] Jadex User Guide, http://prdownloads.sourceforge.net/jadex/userguide-0.96.pdf

[66] Jadex Tool Guide, http://prdownloads.sourceforge.net/jadex/toolguide-0.96.pdf

[67] H. Sofia Pinto, Christoph Tempich, Steffen Staab, and York Sure. Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engingeering of ontologies. In Ramon L´opez de M´antaras and Lorenza Saitta, editors, Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), August 22nd - 27th, pages 393–397, Valencia, Spain, IOS Press, 2004.

[68] Kotis, K. and G.A. Vouros, Human-centered ontology engineering: The HCOME Methodology. Knowledge and Information Systems, 10(1): pp. 109-131, 2005.

[69] Sure, Y., "Methodology, tools and case studies for ontology based knowledge management", http://www.ubka.uni-karlsruhe.de/cgi/bin/psview?document=2003/wiwi/6

[70] Corcho, O., Fernandez-Lopez, M., Gomez-Perez, A., Methodologies, tools, and languages for building ontologies. Where is their meeting point?, Data and Knowledge engineering, 2003.

[71] Uschold, M.: Where are the Semantics in the Semantic Web? AI Magazine, v.24, n.3, p.25-36, September 2003.

[72] Jambalaya: http://www.thechiselgroup.org/jambalaya/

[73] Gruninger, M., and Fox, M.S. (1995), Methodology for the Design and Evaluation of Ontologies, Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95, Montreal,1995.

[74] M. Uschold and M. Grüninger: Ontologies: Principles, methods and applications. Knowledge Engineering Review, 11(2), 1996.

File: smarthouse_upper.owl
- represents the Smart House Upper Ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:time="http://daml.umbc.edu/ontologies/cobra/0.4/calendarclock#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:abstractTime="http://daml.umbc.edu/ontologies/cobra/0.4/time-basic#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://mercur.utcluj.ro/benta/smarthouse/ontology/vs1/smarhouse#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://mercur.utcluj.ro/benta/smarthouse/ontology/vs1/smarhouse">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://daml.umbc.edu/ontologies/cobra/0.4/time-basic"/>
    <owl:imports rdf:resource="http://daml.umbc.edu/ontologies/cobra/0.4/calendarclock"/>
  </owl:Ontology>
  <owl:Class rdf:ID="SystemDate">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Time"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:ID="SystemTime"/>
    </owl:disjointWith>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Day Month Year</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="NonHuman">
    <owl:disjointWith>
      <owl:Class rdf:ID="Human"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Individ"/>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >An animal or a robot.</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Service">
    <owl:disjointWith>
      <owl:Class rdf:ID="Agent"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Network"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Device"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="CompEntity"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Actor">
    <owl:disjointWith>
      <owl:Class rdf:ID="PhysicalObject"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="State"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Time"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Location"/>
    </owl:disjointWith>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >The subject of an action in the context. Ex. the human adult John or the pet robot.</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="ContextEntity"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:ID="Weather"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#CompEntity"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Activity"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#ContextEntity">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >It contains different type of contexst elements</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="#SystemTime">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >hours : minutes</rdfs:comment>
    <owl:disjointWith rdf:resource="#SystemDate"/>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Time"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Mental">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >What's in the Actor's mind :)</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#State"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:ID="Affective"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Physiological"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#CompEntity">
    <owl:disjointWith>
      <owl:Class rdf:about="#PhysicalObject"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Time"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Location"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#Actor"/>
    <rdfs:subClassOf rdf:resource="#ContextEntity"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Every entity in the context able to calculate.</rdfs:comment>
    <owl:disjointWith>
      <owl:Class rdf:about="#Activity"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#State"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Weather"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="ForbiddenActivity">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
```

```
        <owl:Class rdf:about="#Activity"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="isForbiddenTo"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#Actor"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="#Physiological">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#State"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:about="#Affective"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Mental"/>
</owl:Class>
<owl:Class rdf:ID="Group">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >A number of individuals</rdfs:comment>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Actor"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="hasMember"/>
          </owl:onProperty>
          <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >2</owl:minCardinality>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasMember"/>
          </owl:onProperty>
          <owl:allValuesFrom>
            <owl:Class rdf:about="#Individ"/>
          </owl:allValuesFrom>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasMember"/>
          </owl:onProperty>
          <owl:someValuesFrom>
            <owl:Class rdf:about="#Individ"/>
          </owl:someValuesFrom>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="http://daml.umbc.edu/ontologies/cobra/0.4/calendarclock#CalendarDescription">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Time"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="IndoorSpace">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Location"/>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Inside the building.</rdfs:comment>
  <owl:disjointWith>
    <owl:Class rdf:ID="OutdoorSpace"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#Location">
  <owl:disjointWith rdf:resource="#Actor"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#Time"/>
  </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#ContextEntity"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#Weather"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Activity"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#State"/>
  </owl:disjointWith>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The position in the context as names of places. Ex. Bedroom.</rdfs:comment>
  <owl:disjointWith>
    <owl:Class rdf:about="#PhysicalObject"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#CompEntity"/>
</owl:Class>
<owl:Class rdf:about="#Affective">
  <owl:disjointWith rdf:resource="#Mental"/>
  <owl:disjointWith rdf:resource="#Physiological"/>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#State"/>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The emotional state.</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="#Human">
  <owl:disjointWith rdf:resource="#NonHuman"/>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Individ"/>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >A person.</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="NeuralNetwork">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Offers the support for representing weighted relations between the context elements</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="#Device">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Computational device - software and harware capabilities.</rdfs:comment>
  <owl:disjointWith>
    <owl:Class rdf:about="#Agent"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Network"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Service"/>
  <rdfs:subClassOf rdf:resource="#CompEntity"/>
</owl:Class>
<owl:Class rdf:about="#Agent">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Software agent.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#CompEntity"/>
  <owl:disjointWith rdf:resource="#Device"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#Network"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Service"/>
</owl:Class>
```

```
<owl:Class rdf:ID="ScheduledActivity">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Activity"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="hasScheduledTime"/>
          </owl:onProperty>
          <owl:someValuesFrom>
            <owl:Class rdf:about="#Time"/>
          </owl:someValuesFrom>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="isPerformedBy"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#Actor"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >When is his birthday he gives a party.</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="DeducedActivity">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Activity"/>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The Actor's activity can be infered. Ex. he eats.</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="#Time">
  <owl:disjointWith>
    <owl:Class rdf:about="#State"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Weather"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#CompEntity"/>
  <owl:disjointWith rdf:resource="#Actor"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#Activity"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#PhysicalObject"/>
  </owl:disjointWith>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The representation of the measure of time in the context. Ex. 10:00, day 25, Sunday, November, 2007</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#ContextEntity"/>
  <owl:disjointWith rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:about="#Network">
  <owl:disjointWith rdf:resource="#Agent"/>
  <owl:disjointWith rdf:resource="#Device"/>
  <owl:disjointWith rdf:resource="#Service"/>
  <rdfs:subClassOf rdf:resource="#CompEntity"/>
</owl:Class>
<owl:Class rdf:ID="Current">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Affective"/>
        <owl:Class rdf:about="#Physiological"/>
        <owl:Class rdf:about="#Mental"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#State"/>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The current state can have one to all of the affective, mental and physiological states.</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="Sensitivity">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Measure  the Actor's tendency to get into a certain State when presented to a stimulus.</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="#Individ">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Human"/>
        <owl:Class rdf:about="#NonHuman"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#Actor"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >A human or non-human individ. Ex. a robot.</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="Weight">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >To evaluate the importance of an entrance.</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Neuron"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Preference"/>
<owl:Class rdf:about="#Neuron">
  <rdfs:subClassOf rdf:resource="#NeuralNetwork"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The ontological model of neuron's computer model.</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="#OutdoorSpace">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Outside the building</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Location"/>
  <owl:disjointWith rdf:resource="#IndoorSpace"/>
</owl:Class>
<owl:Class rdf:about="#PhysicalObject">
  <owl:disjointWith rdf:resource="#Actor"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#State"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Weather"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#CompEntity"/>
  <owl:disjointWith rdf:resource="#Location"/>
  <owl:disjointWith rdf:resource="#Time"/>
  <rdfs:subClassOf rdf:resource="#ContextEntity"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Every object within the ontology. Ex. the table.</rdfs:comment>
  <owl:disjointWith>
    <owl:Class rdf:about="#Activity"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#Activity">
  <owl:disjointWith rdf:resource="#PhysicalObject"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#State"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Actor"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#Weather"/>
  </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="#ContextEntity"/>
  <owl:disjointWith rdf:resource="#Time"/>
  <owl:disjointWith rdf:resource="#CompEntity"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >What an Actor is engaged in. Ex. sleeping.</rdfs:comment>
  <owl:disjointWith rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:about="#State">
  <owl:disjointWith rdf:resource="#CompEntity"/>
  <owl:disjointWith rdf:resource="#Actor"/>
  <owl:disjointWith rdf:resource="#PhysicalObject"/>
  <owl:disjointWith rdf:resource="#Activity"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#Weather"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Location"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Describes the general situation of an Actor. Ex. happy, hungry, interested.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#ContextEntity"/>
  <owl:disjointWith rdf:resource="#Time"/>
</owl:Class>
<owl:Class rdf:about="#Weather">
  <owl:disjointWith rdf:resource="#Location"/>
  <owl:disjointWith rdf:resource="#PhysicalObject"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The atmosferic conditions expressed by tags: sunny, windy, clouded.</rdfs:comment>
  <owl:disjointWith rdf:resource="#Activity"/>
  <owl:disjointWith rdf:resource="#Actor"/>
  <owl:disjointWith rdf:resource="#CompEntity"/>
  <rdfs:subClassOf rdf:resource="#ContextEntity"/>
  <owl:disjointWith rdf:resource="#State"/>
  <owl:disjointWith rdf:resource="#Time"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="use">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >In an activity the actor can use an object or a computational entity.</rdfs:comment>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#CompEntity"/>
        <owl:Class rdf:about="#PhysicalObject"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Activity"/>
        <owl:Class rdf:about="#Actor"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="sensitivityOfTheState">
  <rdfs:range rdf:resource="#State"/>
  <rdfs:domain rdf:resource="#Sensitivity"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="engagedIn">
  <rdfs:range rdf:resource="#Activity"/>
  <rdfs:domain rdf:resource="#Actor"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >An Actor can be engaged in different actions.Ex: The robot cleans the floor.</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isPerformedBy">
```

```
  <rdfs:domain rdf:resource="#Activity"/>
  <rdfs:range rdf:resource="#Actor"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_before_1">
  <rdfs:subPropertyOf rdf:resource="http://daml.umbc.edu/ontologies/cobra/0.4/time-basic#after"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isNotAllowedToPerform">
  <rdfs:domain rdf:resource="#Actor"/>
  <rdfs:range rdf:resource="#Activity"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isForbiddenTo"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="own">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Actor's property.</rdfs:comment>
  <rdfs:domain rdf:resource="#Actor"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#CompEntity"/>
        <owl:Class rdf:about="#PhysicalObject"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasScheduledTime">
  <rdfs:domain rdf:resource="#Activity"/>
  <rdfs:range rdf:resource="#Time"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="IsMemberOf">
  <rdfs:domain rdf:resource="#Individ"/>
  <rdfs:range rdf:resource="#Group"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasMember"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isForbiddenTo">
  <owl:inverseOf rdf:resource="#isNotAllowedToPerform"/>
  <rdfs:range rdf:resource="#Actor"/>
  <rdfs:domain rdf:resource="#Activity"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isInTheState">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The Actor is in the state. Ex. John is happy.</rdfs:comment>
  <rdfs:range rdf:resource="#Current"/>
  <rdfs:domain rdf:resource="#Actor"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasSensitivity">
  <rdfs:range rdf:resource="#Sensitivity"/>
  <rdfs:domain rdf:resource="#Actor"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The Actor can have sensitivity.</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isOfStateType">
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Affective"/>
        <owl:Class rdf:about="#Mental"/>
        <owl:Class rdf:about="#Physiological"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdfs:domain rdf:resource="#Current"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The current state can be one of affective, mental or physiological.</rdfs:comment>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="induces">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >A context entity can induce a certain state. Ex. When time is up John (an Human Actor) can feels
liberated.</rdfs:comment>
    <rdfs:domain rdf:resource="#ContextEntity"/>
    <rdfs:range rdf:resource="#State"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasMember">
    <rdfs:range rdf:resource="#Individ"/>
    <rdfs:domain rdf:resource="#Group"/>
    <owl:inverseOf rdf:resource="#IsMemberOf"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="sensitivityValue">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >The sensitivity values are of float type.</rdfs:comment>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdfs:domain rdf:resource="#Sensitivity"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hasAge">
    <rdfs:domain rdf:resource="#Human"/>
  </owl:DatatypeProperty>
  <owl:TransitiveProperty rdf:ID="isLocatedIn">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Actors and objects can change position in the context.</rdfs:comment>
    <rdfs:range rdf:resource="#Location"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Actor"/>
          <owl:Class rdf:about="#PhysicalObject"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <owl:inverseOf>
      <owl:TransitiveProperty rdf:ID="spatialContains"/>
    </owl:inverseOf>
  </owl:TransitiveProperty>
  <owl:TransitiveProperty rdf:about="#spatialContains">
    <rdfs:domain rdf:resource="#Location"/>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#PhysicalObject"/>
          <owl:Class rdf:about="#Actor"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <owl:inverseOf rdf:resource="#isLocatedIn"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >What we may find at a location.</rdfs:comment>
  </owl:TransitiveProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.4, Build 126)  http://protege.stanford.edu -->
```

Jadex Classes for inter-agent communication

```
/* */

/*
 * $class.javaName.java
 *
 * Generated by Protege plugin Beanynizer.
 * This class implements the functionality of Sensor.
 * Feel free to change.
 *
 */
package ontology.onto;


/**
 *  Editable Java class for concept <code>Sensor</code> of SmartHouseOntology ontology.
 */
public class Sensor  extends SensorData implements nuggets.INugget
{
	//-------- constructors --------

	/**
	 *  Default Constructor. <br>
	 *  Create a new <code>Sensor</code>.
	 */
	public Sensor() {
		// Empty constructor required for JavaBeans (do not remove).
	}

	/**
	 *  Init Constructor. <br>
	 *  Create a new <code>Sensor</code>.
	 * @param devicestatus
	 * @param haslocation
	 * @param instancename
	 * @param sensorvalue
	 * @param timestamp
	 */
	public Sensor(String devicestatus, Location haslocation, String instancename, String sensorvalue, String timestamp)
{
		// Constructor using required slots (change if desired).
		setDeviceStatus(devicestatus);
		setHasLocation(haslocation);
		setInstanceName(instancename);
		setSensorValue(sensorvalue);
		setTimestamp(timestamp);
	}


	/**
	 *  Clone Constructor. <br>
	 *  Create a new <code>Sensor</code>.<br>
	 *  Copy all attributes from <code>proto</code> to this instance.
	 *
	 *  @param proto The prototype instance.
	 */
	public Sensor(Sensor proto) {
		setDeviceStatus(proto.getDeviceStatus());
		setHasLocation(proto.getHasLocation());
		setInstanceName(proto.getInstanceName());
		setSensorValue(proto.getSensorValue());
		setTimestamp(proto.getTimestamp());
	}

	//-------- custom code --------
	//-------- Object methods -----

	/**
	 *  Get a string representation of this <code>Sensor</code>.
	 *  @return The string representation.
	 */
	public String toString() {
		return "Sensor("
		+ "devicestatus="+getDeviceStatus()
		+ ", haslocation="+getHasLocation()
		+ ", instancename="+getInstanceName()
		+ ", sensorvalue="+getSensorValue()
		+ ", timestamp="+getTimestamp()
		+ ")";
	}

	/**
	 *  Get a clone of this <code>Sensor</code>.
	 *  @return a shalow copy of this instance.
	 */
	public Object clone() {
		return new Sensor(this);
	}

	/**
	 *  Test the equality of this <code>Sensor</code>
	 *  and an object <code>obj</code>.
	 *
	 *  @param obj the object this test will be performed with
	 *  @return false if <code>obj</code> is not of <code>Sensor</code> class,
	 *         true if all attributes are equal.
	 */
	public boolean equals(Object obj) {
		if (obj instanceof Sensor) {
			Sensor cmp=(Sensor)obj;
			if (getDeviceStatus()!=cmp.getDeviceStatus() &&
					(getDeviceStatus()==null
|| !getDeviceStatus().equals(cmp.getDeviceStatus()))
			) return false;
			if (getHasLocation()!=cmp.getHasLocation() &&
					(getHasLocation()==null
|| !getHasLocation().equals(cmp.getHasLocation()))
			) return false;
			if (getInstanceName()!=cmp.getInstanceName() &&
					(getInstanceName()==null
|| !getInstanceName().equals(cmp.getInstanceName()))
			) return false;
			if (getSensorValue()!=cmp.getSensorValue() &&
					(getSensorValue()==null
|| !getSensorValue().equals(cmp.getSensorValue()))
			) return false;
			if (getTimestamp()!=cmp.getTimestamp() &&
					(getTimestamp()==null || !getTimestamp().equals(cmp.getTimestamp()))
			) return false;
			return true;
		}
		return false;
	}
}

/*
 * SensorData.java
 *
 * Generated by Protege plugin Beanynizer.
 * Changes will be lost!
 */
package ontology.onto;



/**
 *  Java class for concept Sensor of SmartHouseOntology ontology.
 */
public abstract class SensorData		extends Device implements nuggets.INugget
{
	//-------- constants ----------

	/** Predefined value "LOW" for slot sensorValue. */
	public static String  SENSORVALUE_LOW  = "LOW";
	/** Predefined value "MEDIUM" for slot sensorValue. */
	public static String  SENSORVALUE_MEDIUM  = "MEDIUM";
	/** Predefined value "HIGH" for slot sensorValue. */
	public static String  SENSORVALUE_HIGH  = "HIGH";

	//-------- attributes ----------

	/** Attribute for slot sensorValue. */
	protected  String  sensorvalue;

	//-------- constructors --------

	/**
	 *  Default Constructor. <br>
	 *  Create a new <code>Sensor</code>.
	 */
	public SensorData()  { //
	}

	//-------- accessor methods --------

	/**
	 *  Get the sensorValue of this Sensor.
	 * @return sensorValue
	 */
	public String  getSensorValue() {
		return this.sensorvalue;
	}

	/**
	 *  Set the sensorValue of this Sensor.
	 * @param sensorvalue the value to be set
	 */
	public void  setSensorValue(String sensorvalue) {
		this.sensorvalue = sensorvalue;
	}

	//-------- object methods --------

	/**
	 *  Get a string representation of this Sensor.
	 *  @return The string representation.
	 */
	public String toString() {
		return "Sensor("
		+ "devicestatus="+getDeviceStatus()
		+ ", haslocation="+getHasLocation()
		+ ", instancename="+getInstanceName()
		+ ", sensorvalue="+getSensorValue()
		+ ", timestamp="+getTimestamp()
		+ ")";
	}
	//--------- nuggets methods ---------

	/**
	 * Persist this Sensor using the nuggets utility.
	 * @param c
	 */
	public void _persist(nuggets.ICruncher c) {
		// declare references
		int idHasLocation = c.declare(haslocation);
		// persist the nugget
		c.startConcept(this);
		if (devicestatus!=null)
			c.put("devicestatus", devicestatus);
		if (idHasLocation>0)
			c.put("haslocation", idHasLocation);
		if (instancename!=null)
			c.put("instancename", instancename);
		if (sensorvalue!=null)
			c.put("sensorvalue", sensorvalue);
		if (timestamp!=null)
			c.put("timestamp", timestamp);
	}

	/**
	 * Restore this Sensor
	 * @param a the name of the attribute
	 * @param v the value of the attribute
	 */
	public void _set(String a, Object v) { //
		switch(hash(a)) {
		case 0: devicestatus =  (String)v; return;
		case 1: timestamp =  (String)v; return;
		case 2: haslocation =  (Location)v; return;
		case 3: sensorvalue =  (String)v; return;
		case 4: instancename =  (String)v; return;
		}
	}

	private static final int hash(String name) {
		return ((714909982*name.charAt(0))>>>15)%5;
	}

}

/*
 * $class.javaName.java
 *
 * Generated by Protege plugin Beanynizer.
 * This class implements the functionality of TemperatureSensor.
 * Feel free to change.
 *
 */
package ontology.onto;


/**
 *  Editable Java class for concept <code>TemperatureSensor</code> of SmartHouseOntology ontology.
 */
public class TemperatureSensor  extends TemperatureSensorData implements nuggets.INugget
{
	//-------- constructors --------

	/**
	 *  Default Constructor. <br>
	 *  Create a new <code>TemperatureSensor</code>.
	 */
	public TemperatureSensor() {
		// Empty constructor required for JavaBeans (do not remove).
	}

	/**
	 *  Init Constructor. <br>
	 *  Create a new <code>TemperatureSensor</code>.
	 * @param devicestatus
	 * @param haslocation
	 * @param instancename
	 * @param sensorvalue
	 * @param timestamp
	 */
	public TemperatureSensor(String devicestatus, Location haslocation, String instancename, String sensorvalue,
String timestamp) {
		// Constructor using required slots (change if desired).
		setDeviceStatus(devicestatus);
		setHasLocation(haslocation);
		setInstanceName(instancename);
		setSensorValue(sensorvalue);
		setTimestamp(timestamp);
	}


	/**
	 *  Clone Constructor. <br>
	 *  Create a new <code>TemperatureSensor</code>.<br>
	 *  Copy all attributes from <code>proto</code> to this instance.
	 *
	 *  @param proto The prototype instance.
	 */
	public TemperatureSensor(TemperatureSensor proto) {
		setDeviceStatus(proto.getDeviceStatus());
		setHasLocation(proto.getHasLocation());
		setInstanceName(proto.getInstanceName());
		setSensorValue(proto.getSensorValue());
		setTimestamp(proto.getTimestamp());
	}

	//-------- custom code --------


	//-------- Object methods -----

	/**
	 *  Get a string representation of this <code>TemperatureSensor</code>.
	 *  @return The string representation.
	 */
	public String toString() {
		return "TemperatureSensor("
		+ "devicestatus="+getDeviceStatus()
		+ ", haslocation="+getHasLocation()
		+ ", instancename="+getInstanceName()
		+ ", sensorvalue="+getSensorValue()
		+ ", timestamp="+getTimestamp()
		+ ")";
	}

	/**
	 *  Get a clone of this <code>TemperatureSensor</code>.
	 *  @return a shalow copy of this instance.
	 */
	public Object clone() {
		return new TemperatureSensor(this);
	}

	/**
	 *  Test the equality of this <code>TemperatureSensor</code>
	 *  and an object <code>obj</code>.
	 *
	 *  @param obj the object this test will be performed with
	 *  @return false if <code>obj</code> is not of <code>TemperatureSensor</code> class,
	 *  	true if all attributes are equal.
	 */
	public boolean equals(Object obj) {
		if (obj instanceof TemperatureSensor) {
			TemperatureSensor cmp=(TemperatureSensor)obj;
			if (getDeviceStatus()!=cmp.getDeviceStatus() &&
					(getDeviceStatus()==null
|| !getDeviceStatus().equals(cmp.getDeviceStatus()))
			) return false;
			if (getHasLocation()!=cmp.getHasLocation() &&
					(getHasLocation()==null
|| !getHasLocation().equals(cmp.getHasLocation()))
			) return false;
			if (getInstanceName()!=cmp.getInstanceName() &&
					(getInstanceName()==null
|| !getInstanceName().equals(cmp.getInstanceName()))
			) return false;
			if (getSensorValue()!=cmp.getSensorValue() &&
					(getSensorValue()==null
|| !getSensorValue().equals(cmp.getSensorValue()))
			) return false;
			if (getTimestamp()!=cmp.getTimestamp() &&
					(getTimestamp()==null || !getTimestamp().equals(cmp.getTimestamp()))
			) return false;
			return true;
		}
		return false;
	}
}


/*
 * TemperatureSensorData.java
 *
 * Generated by Protege plugin Beanynizer.
 * Changes will be lost!
 */
package ontology.onto;



/**
 *  Java class for concept TemperatureSensor of SmartHouseOntology ontology.
 */
public abstract class TemperatureSensorData	extends Sensor implements nuggets.INugget
{
	//-------- constants ----------

	//-------- attributes ----------

	//-------- constructors --------

	/**
	 *  Default Constructor. <br>
	 *  Create a new <code>TemperatureSensor</code>.
	 */
	public TemperatureSensorData()  { //
	}

	//-------- accessor methods --------

	//-------- object methods --------
```

```
		/**
		 *  Get a string representation of this TemperatureSensor.
		 *  @return The string representation.
		 */
		public String toString() {
			return "TemperatureSensor("
			+ "devicestatus="+getDeviceStatus()
			+ ", haslocation="+getHasLocation()
			+ ", instancename="+getInstanceName()
			+ ", sensorvalue="+getSensorValue()
			+ ", timestamp="+getTimestamp()
			+ ")";
		}

		//--------- nuggets methods ---------

		/**
		 * Persist this TemperatureSensor using the nuggets utility.
		 * @param c
		 */
		public void _persist(nuggets.ICruncher c) {
			// declare references
			int idHasLocation = c.declare(haslocation);
			// persist the nugget
			c.startConcept(this);
			if (devicestatus!=null)
				c.put("devicestatus", devicestatus);
			if (idHasLocation>0)
				c.put("haslocation", idHasLocation);
			if (instancename!=null)
				c.put("instancename", instancename);
			if (sensorvalue!=null)
				c.put("sensorvalue", sensorvalue);
			if (timestamp!=null)
				c.put("timestamp", timestamp);
		}

		/**
		 * Restore this TemperatureSensor
		 * @param a the name of the attribute
		 * @param v the value of the attribute
		 */
		public void _set(String a, Object v) { //
			switch(hash(a)) {
			case 0: sensorvalue =  (String)v; return;
			case 1: haslocation =  (Location)v; return;
			case 2: devicestatus =  (String)v; return;
			case 3: timestamp =  (String)v; return;
			case 4: instancename =  (String)v; return;
			}
		}

		private static final int hash(String name) {
			return ((1559761670*name.charAt(0))>>>15)%5;
		}

}

/*
 * $class.javaName.java
 *
 * Generated by Protege plugin Beanynizer.
 * This class implements the functionality of LightSensor.
 * Feel free to change.
 *
 */
package ontology.onto;
```

```
/**
 *  Editable Java class for concept <code>LightSensor</code> of SmartHouseOntology ontology.
 */
public class LightSensor  extends LightSensorData implements nuggets.INugget
{
	//-------- constructors --------

	/**
	 *  Default Constructor. <br>
	 *  Create a new <code>LightSensor</code>.
	 */
	public LightSensor() {
		// Empty constructor required for JavaBeans (do not remove).
	}

	/**
	 *  Init Constructor. <br>
	 *  Create a new <code>LightSensor</code>.
	 * @param devicestatus
	 * @param haslocation
	 * @param instancename
	 * @param sensorvalue
	 * @param timestamp
	 */
	public LightSensor(String devicestatus, Location haslocation, String instancename, String sensorvalue, String 
timestamp) {
		// Constructor using required slots (change if desired).
		setDeviceStatus(devicestatus);
		setHasLocation(haslocation);
		setInstanceName(instancename);
		setSensorValue(sensorvalue);
		setTimestamp(timestamp);
	}


	/**
	 *  Clone Constructor. <br>
	 *  Create a new <code>LightSensor</code>.<br>
	 *  Copy all attributes from <code>proto</code> to this instance.
	 *
	 *  @param proto The prototype instance.
	 */
	public LightSensor(LightSensor proto) {
		setDeviceStatus(proto.getDeviceStatus());
		setHasLocation(proto.getHasLocation());
		setInstanceName(proto.getInstanceName());
		setSensorValue(proto.getSensorValue());
		setTimestamp(proto.getTimestamp());
	}

	//-------- custom code --------


	//-------- Object methods -----

	/**
	 *  Get a string representation of this <code>LightSensor</code>.
	 *  @return The string representation.
	 */
	public String toString() {
		return "LightSensor("
		+ "devicestatus="+getDeviceStatus()
		+ ", haslocation="+getHasLocation()
		+ ", instancename="+getInstanceName()
		+ ", sensorvalue="+getSensorValue()
```

```
                    + ", timestamp="+getTimestamp()
                    + ")";
          }

          /**
           *  Get a clone of this <code>LightSensor</code>.
           *  @return a shalow copy of this instance.
           */
          public Object clone() {
                    return new LightSensor(this);
          }

          /**
           *  Test the equality of this <code>LightSensor</code>
           *  and an object <code>obj</code>.
           *
           *  @param obj the object this test will be performed with
           *  @return false if <code>obj</code> is not of <code>LightSensor</code> class,
           *         true if all attributes are equal.
           */
          public boolean equals(Object obj) {
                    if (obj instanceof LightSensor) {
                              LightSensor cmp=(LightSensor)obj;
                              if (getDeviceStatus()!=cmp.getDeviceStatus() &&
                                                  (getDeviceStatus()==null
|| !getDeviceStatus().equals(cmp.getDeviceStatus()))
                              ) return false;
                              if (getHasLocation()!=cmp.getHasLocation() &&
                                                  (getHasLocation()==null
|| !getHasLocation().equals(cmp.getHasLocation()))
                              ) return false;
                              if (getInstanceName()!=cmp.getInstanceName() &&
                                                  (getInstanceName()==null
|| !getInstanceName().equals(cmp.getInstanceName()))
                              ) return false;
                              if (getSensorValue()!=cmp.getSensorValue() &&
                                                  (getSensorValue()==null
|| !getSensorValue().equals(cmp.getSensorValue()))
                              ) return false;
                              if (getTimestamp()!=cmp.getTimestamp() &&
                                                  (getTimestamp()==null || !getTimestamp().equals(cmp.getTimestamp()))
                              ) return false;
                              return true;
                    }
                    return false;
          }
}


/*
 * LightSensorData.java
 *
 * Generated by Protege plugin Beanynizer.
 * Changes will be lost!
 */
package ontology.onto;



/**
 *  Java class for concept LightSensor of SmartHouseOntology ontology.
 */
public abstract class LightSensorData    extends Sensor implements nuggets.INugget
{
          //-------- constants ----------

          //-------- attributes ----------
```

```
//-------- constructors --------

/**
 *  Default Constructor. <br>
 *  Create a new <code>LightSensor</code>.
 */
public LightSensorData()  { //
}

//-------- accessor methods --------

//-------- object methods --------

/**
 *  Get a string representation of this LightSensor.
 *  @return The string representation.
 */
public String toString() {
		return "LightSensor("
		+ "devicestatus="+getDeviceStatus()
		+ ", haslocation="+getHasLocation()
		+ ", instancename="+getInstanceName()
		+ ", sensorvalue="+getSensorValue()
		+ ", timestamp="+getTimestamp()
		+ ")";
}

//--------- nuggets methods ---------

/**
 * Persist this LightSensor using the nuggets utility.
 * @param c
 */
public void _persist(nuggets.ICruncher c) {
		// declare references
		int idHasLocation = c.declare(haslocation);
		// persist the nugget
		c.startConcept(this);
		if (devicestatus!=null)
				c.put("devicestatus", devicestatus);
		if (idHasLocation>0)
				c.put("haslocation", idHasLocation);
		if (instancename!=null)
				c.put("instancename", instancename);
		if (sensorvalue!=null)
				c.put("sensorvalue", sensorvalue);
		if (timestamp!=null)
				c.put("timestamp", timestamp);
}

/**
 * Restore this LightSensor
 * @param a the name of the attribute
 * @param v the value of the attribute
 */
public void _set(String a, Object v) { //
		switch(hash(a)) {
		case 0: sensorvalue =  (String)v; return;
		case 1: timestamp =  (String)v; return;
		case 2: haslocation =  (Location)v; return;
		case 3: instancename =  (String)v; return;
		case 4: devicestatus =  (String)v; return;
		}
}

private static final int hash(String name) {
		return ((1050256105*name.charAt(0))>>>15)%5;
	}

}

/*
 * $class.javaName.java
 *
 * Generated by Protege plugin Beanynizer.
 * This class implements the functionality of HumiditySensor.
 * Feel free to change.
 *
 */
package ontology.onto;


/**
 *  Editable Java class for concept <code>HumiditySensor</code> of SmartHouseOntology ontology.
 */
public class HumiditySensor  extends HumiditySensorData implements nuggets.INugget
{
	//-------- constructors --------

	/**
	 *  Default Constructor. <br>
	 *  Create a new <code>HumiditySensor</code>.
	 */
	public HumiditySensor() {
		// Empty constructor required for JavaBeans (do not remove).
	}

	/**
	 *  Init Constructor. <br>
	 *  Create a new <code>HumiditySensor</code>.
	 * @param devicestatus
	 * @param haslocation
	 * @param instancename
	 * @param sensorvalue
	 * @param timestamp
	 */
	public HumiditySensor(String devicestatus, Location haslocation, String instancename, String sensorvalue, String
timestamp) {
		// Constructor using required slots (change if desired).
		setDeviceStatus(devicestatus);
		setHasLocation(haslocation);
		setInstanceName(instancename);
		setSensorValue(sensorvalue);
		setTimestamp(timestamp);
	}


	/**
	 *  Clone Constructor. <br>
	 *  Create a new <code>HumiditySensor</code>.<br>
	 *  Copy all attributes from <code>proto</code> to this instance.
	 *
	 *  @param proto The prototype instance.
	 */
	public HumiditySensor(HumiditySensor proto) {
		setDeviceStatus(proto.getDeviceStatus());
		setHasLocation(proto.getHasLocation());
		setInstanceName(proto.getInstanceName());
		setSensorValue(proto.getSensorValue());
		setTimestamp(proto.getTimestamp());
	}

	//-------- custom code --------
	//-------- Object methods -----

	/**
	 *  Get a string representation of this <code>HumiditySensor</code>.
	 *  @return The string representation.
	 */
	public String toString() {
		return "HumiditySensor("
		+ "devicestatus="+getDeviceStatus()
		+ ", haslocation="+getHasLocation()
		+ ", instancename="+getInstanceName()
		+ ", sensorvalue="+getSensorValue()
		+ ", timestamp="+getTimestamp()
		+ ")";
	}

	/**
	 *  Get a clone of this <code>HumiditySensor</code>.
	 *  @return a shalow copy of this instance.
	 */
	public Object clone() {
		return new HumiditySensor(this);
	}

	/**
	 *  Test the equality of this <code>HumiditySensor</code>
	 *  and an object <code>obj</code>.
	 *
	 *  @param obj the object this test will be performed with
	 *  @return false if <code>obj</code> is not of <code>HumiditySensor</code> class,
	 *         true if all attributes are equal.
	 */
	public boolean equals(Object obj) {
		if (obj instanceof HumiditySensor) {
			HumiditySensor cmp=(HumiditySensor)obj;
			if (getDeviceStatus()!=cmp.getDeviceStatus() &&
					(getDeviceStatus()==null
|| !getDeviceStatus().equals(cmp.getDeviceStatus()))
			) return false;
			if (getHasLocation()!=cmp.getHasLocation() &&
					(getHasLocation()==null
|| !getHasLocation().equals(cmp.getHasLocation()))
			) return false;
			if (getInstanceName()!=cmp.getInstanceName() &&
					(getInstanceName()==null
|| !getInstanceName().equals(cmp.getInstanceName()))
			) return false;
			if (getSensorValue()!=cmp.getSensorValue() &&
					(getSensorValue()==null
|| !getSensorValue().equals(cmp.getSensorValue()))
			) return false;
			if (getTimestamp()!=cmp.getTimestamp() &&
					(getTimestamp()==null || !getTimestamp().equals(cmp.getTimestamp()))
			) return false;
			return true;
		}
		return false;
	}
}


/*
 * HumiditySensorData.java
 *
 * Generated by Protege plugin Beanynizer.
 * Changes will be lost!
 */
package ontology.onto;



/**
 *  Java class for concept HumiditySensor of SmartHouseOntology ontology.
 */
public abstract class HumiditySensorData		extends Sensor implements nuggets.INugget
{
	//-------- constants ----------

	//-------- attributes ----------

	//-------- constructors --------

	/**
	 *  Default Constructor. <br>
	 *  Create a new <code>HumiditySensor</code>.
	 */
	public HumiditySensorData()  { //
	}

	//-------- accessor methods --------

	//-------- object methods --------

	/**
	 *  Get a string representation of this HumiditySensor.
	 *  @return The string representation.
	 */
	public String toString() {
		return "HumiditySensor("
		+ "devicestatus="+getDeviceStatus()
		+ ", haslocation="+getHasLocation()
		+ ", instancename="+getInstanceName()
		+ ", sensorvalue="+getSensorValue()
		+ ", timestamp="+getTimestamp()
		+ ")";
	}

	//--------- nuggets methods ---------

	/**
	 * Persist this HumiditySensor using the nuggets utility.
	 * @param c
	 */
	public void _persist(nuggets.ICruncher c) {
		// declare references
		int idHasLocation = c.declare(haslocation);
		// persist the nugget
		c.startConcept(this);
		if (devicestatus!=null)
			c.put("devicestatus", devicestatus);
		if (idHasLocation>0)
			c.put("haslocation", idHasLocation);
		if (instancename!=null)
			c.put("instancename", instancename);
		if (sensorvalue!=null)
			c.put("sensorvalue", sensorvalue);
		if (timestamp!=null)
			c.put("timestamp", timestamp);
	}

	/**
	 * Restore this HumiditySensor
	 * @param a the name of the attribute
	 * @param v the value of the attribute
	 */
	public void _set(String a, Object v) { //
		switch(hash(a)) {
		case 0: devicestatus =  (String)v; return;
		case 1: sensorvalue =  (String)v; return;
		case 2: haslocation =  (Location)v; return;
		case 3: instancename =  (String)v; return;
		case 4: timestamp =  (String)v; return;
		}
	}

	private static final int hash(String name) {
		return ((2064133728*name.charAt(0))>>>15)%5;
	}

}


/*
 * $class.javaName.java
 *
 * Generated by Protege plugin Beanynizer.
 * This class implements the functionality of Activity.
 * Feel free to change.
 *
 */
package ontology.onto;


/**
 *  Editable Java class for concept <code>Activity</code> of SmartHouseOntology ontology.
 */
public class Activity  extends ActivityData implements nuggets.INugget
{
	//-------- constructors --------

	/**
	 *  Default Constructor. <br>
	 *  Create a new <code>Activity</code>.
	 */
	public Activity() {
		// Empty constructor required for JavaBeans (do not remove).
	}

	/**
	 *  Init Constructor. <br>
	 *  Create a new <code>Activity</code>.
	 * @param haslocation
	 * @param instancename
	 */
	public Activity(Location haslocation, String instancename) {
		// Constructor using required slots (change if desired).
		setHasLocation(haslocation);
		setInstanceName(instancename);
	}


	/**
	 *  Clone Constructor. <br>
	 *  Create a new <code>Activity</code>.<br>
	 *  Copy all attributes from <code>proto</code> to this instance.
	 *
	 *  @param proto The prototype instance.
	 */
	public Activity(Activity proto) {
			setHasLocation(proto.getHasLocation());
			setInstanceName(proto.getInstanceName());
		}

		//-------- custom code --------


		//-------- Object methods -----

		/**
		 *  Get a string representation of this <code>Activity</code>.
		 *  @return The string representation.
		 */
		public String toString() {
			return "Activity("
			+ "haslocation="+getHasLocation()
			+ ", instancename="+getInstanceName()
			+ ")";
		}

		/**
		 *  Get a clone of this <code>Activity</code>.
		 *  @return a shalow copy of this instance.
		 */
		public Object clone() {
			return new Activity(this);
		}

		/**
		 *  Test the equality of this <code>Activity</code>
		 *  and an object <code>obj</code>.
		 *
		 *  @param obj the object this test will be performed with
		 *  @return false if <code>obj</code> is not of <code>Activity</code> class,
		 *	   true if all attributes are equal.
		 */
		public boolean equals(Object obj) {
			if (obj instanceof Activity) {
				Activity cmp=(Activity)obj;
				if (getHasLocation()!=cmp.getHasLocation() &&
						(getHasLocation()==null
|| !getHasLocation().equals(cmp.getHasLocation()))
				) return false;
				if (getInstanceName()!=cmp.getInstanceName() &&
						(getInstanceName()==null
|| !getInstanceName().equals(cmp.getInstanceName()))
				) return false;
				return true;
			}
			return false;
		}
}

/*
 * ActivityData.java
 *
 * Generated by Protege plugin Beanynizer.
 * Changes will be lost!
 */
package ontology.onto;



/**
 *  Java class for concept Activity of SmartHouseOntology ontology.
 */
public abstract class ActivityData	extends ContextEntity implements nuggets.INugget
```

```
{
        //-------- constants ----------

        //-------- attributes ----------

        /** Attribute for slot hasLocation. */
        protected  Location  haslocation;

        //-------- constructors --------

        /**
         *  Default Constructor. <br>
         *  Create a new <code>Activity</code>.
         */
        public ActivityData()  { //
        }

        //-------- accessor methods --------

        /**
         *  Get the hasLocation of this Activity.
         * @return hasLocation
         */
        public Location  getHasLocation() {
                return this.haslocation;
        }

        /**
         *  Set the hasLocation of this Activity.
         * @param haslocation the value to be set
         */
        public void  setHasLocation(Location haslocation) {
                this.haslocation = haslocation;
        }

        //-------- object methods --------

        /**
         *  Get a string representation of this Activity.
         *  @return The string representation.
         */
        public String toString() {
                return "Activity("
                + "haslocation="+getHasLocation()
                + ", instancename="+getInstanceName()
                + ")";
        }

        //--------- nuggets methods ---------

        /**
         * Persist this Activity using the nuggets utility.
         * @param c
         */
        public void _persist(nuggets.ICruncher c) {
                // declare references
                int idHasLocation = c.declare(haslocation);
                // persist the nugget
                c.startConcept(this);
                if (idHasLocation>0)
                        c.put("haslocation", idHasLocation);
                if (instancename!=null)
                        c.put("instancename", instancename);
        }

        /**
         * Restore this Activity
```

```
	 * @param a the name of the attribute
	 * @param v the value of the attribute
	 */
	public void _set(String a, Object v) { //
		switch(hash(a)) {
		case 0: haslocation =  (Location)v; return;
		case 1: instancename =  (String)v; return;
		}
	}

	private static final int hash(String name) {
		return ((518907758*name.charAt(0))>>>15)%2;
	}

}

/*
 * $class.javaName.java
 *
 * Generated by Protege plugin Beanynizer.
 * This class implements the functionality of Actor.
 * Feel free to change.
 *
 */
package ontology.onto;


/**
 *  Editable Java class for concept <code>Actor</code> of SmartHouseOntology ontology.
 */
public class Actor  extends ActorData implements nuggets.INugget
{
	//-------- constructors --------

	/**
	 *  Default Constructor. <br>
	 *  Create a new <code>Actor</code>.
	 */
	public Actor() {
		// Empty constructor required for JavaBeans (do not remove).
	}

	/**
	 *  Init Constructor. <br>
	 *  Create a new <code>Actor</code>.
	 * @param haslocation
	 * @param instancename
	 */
	public Actor(Location haslocation, String instancename) {
		// Constructor using required slots (change if desired).
		setHasLocation(haslocation);
		setInstanceName(instancename);
	}


	/**
	 *  Clone Constructor. <br>
	 *  Create a new <code>Actor</code>.<br>
	 *  Copy all attributes from <code>proto</code> to this instance.
	 *
	 *  @param proto The prototype instance.
	 */
	public Actor(Actor proto) {
		setHasLocation(proto.getHasLocation());
		setInstanceName(proto.getInstanceName());
	}
	//-------- custom code --------


	//-------- Object methods -----

	/**
	 *  Get a string representation of this <code>Actor</code>.
	 *  @return The string representation.
	 */
	public String toString() {
		return "Actor("
		+ "haslocation="+getHasLocation()
		+ ", instancename="+getInstanceName()
		+ ")";
	}

	/**
	 *  Get a clone of this <code>Actor</code>.
	 *  @return a shalow copy of this instance.
	 */
	public Object clone() {
		return new Actor(this);
	}

	/**
	 *  Test the equality of this <code>Actor</code>
	 *  and an object <code>obj</code>.
	 *
	 *  @param obj the object this test will be performed with
	 *  @return false if <code>obj</code> is not of <code>Actor</code> class,
	 *          true if all attributes are equal.
	 */
	public boolean equals(Object obj) {
		if (obj instanceof Actor) {
			Actor cmp=(Actor)obj;
			if (getHasLocation()!=cmp.getHasLocation() &&
					(getHasLocation()==null
|| !getHasLocation().equals(cmp.getHasLocation()))
			) return false;
			if (getInstanceName()!=cmp.getInstanceName() &&
					(getInstanceName()==null
|| !getInstanceName().equals(cmp.getInstanceName()))
			) return false;
			return true;
		}
		return false;
	}
}

/*
 * ActorData.java
 *
 * Generated by Protege plugin Beanynizer.
 * Changes will be lost!
 */
package ontology.onto;



/**
 *  Java class for concept Actor of SmartHouseOntology ontology.
 */
public abstract class ActorData		extends ContextEntity implements nuggets.INugget
{
	//-------- constants ----------

	//-------- attributes ----------
```

```
/** Attribute for slot hasLocation. */
protected  Location  haslocation;

//-------- constructors --------

/**
 *  Default Constructor. <br>
 *  Create a new <code>Actor</code>.
 */
public ActorData()  { //
}

//-------- accessor methods --------

/**
 *  Get the hasLocation of this Actor.
 * @return hasLocation
 */
public Location  getHasLocation() {
            return this.haslocation;
}

/**
 *  Set the hasLocation of this Actor.
 * @param haslocation the value to be set
 */
public void  setHasLocation(Location haslocation) {
            this.haslocation = haslocation;
}

//-------- object methods --------

/**
 *  Get a string representation of this Actor.
 *  @return The string representation.
 */
public String toString() {
            return "Actor("
            + "haslocation="+getHasLocation()
            + ", instancename="+getInstanceName()
            + ")";
}

//--------- nuggets methods ---------

/**
 * Persist this Actor using the nuggets utility.
 * @param c
 */
public void _persist(nuggets.ICruncher c) {
            // declare references
            int idHasLocation = c.declare(haslocation);
            // persist the nugget
            c.startConcept(this);
            if (idHasLocation>0)
                        c.put("haslocation", idHasLocation);
            if (instancename!=null)
                        c.put("instancename", instancename);
}

/**
 * Restore this Actor
 * @param a the name of the attribute
 * @param v the value of the attribute
 */
public void _set(String a, Object v) { //
```

```
                switch(hash(a)) {
                case 0: haslocation =  (Location)v; return;
                case 1: instancename =  (String)v; return;
                }
        }

        private static final int hash(String name) {
                return ((1387839504*name.charAt(0))>>>15)%2;
        }

}
/*
 * $class.javaName.java
 *
 * Generated by Protege plugin Beanynizer.
 * This class implements the functionality of Device.
 * Feel free to change.
 *
 */
package ontology.onto;


/**
 *  Editable Java class for concept <code>Device</code> of SmartHouseOntology ontology.
 */
public class Device  extends DeviceData implements nuggets.INugget
{
  //-------- constructors --------

  /**
   *  Default Constructor. <br>
   *  Create a new <code>Device</code>.
   */
  public Device() {
    // Empty constructor required for JavaBeans (do not remove).
  }

  /**
   *  Init Constructor. <br>
   *  Create a new <code>Device</code>.
   * @param devicestatus
   * @param haslocation
   * @param instancename
   * @param timestamp
   */
  public Device(String devicestatus, Location haslocation, String instancename, String timestamp) {
    // Constructor using required slots (change if desired).
    setDeviceStatus(devicestatus);
    setHasLocation(haslocation);
    setInstanceName(instancename);
    setTimestamp(timestamp);
  }


  /**
   *  Clone Constructor. <br>
   *  Create a new <code>Device</code>.<br>
   *  Copy all attributes from <code>proto</code> to this instance.
   *
   *  @param proto The prototype instance.
   */
  public Device(Device proto) {
    setDeviceStatus(proto.getDeviceStatus());
    setHasLocation(proto.getHasLocation());
    setInstanceName(proto.getInstanceName());
    setTimestamp(proto.getTimestamp());
  }
  //-------- custom code --------


  //-------- Object methods -----

  /**
   *  Get a string representation of this <code>Device</code>.
   *  @return The string representation.
   */
  public String toString() {
    return "Device("
         + "devicestatus="+getDeviceStatus()
         + ", haslocation="+getHasLocation()
         + ", instancename="+getInstanceName()
         + ", timestamp="+getTimestamp()
         + ")";
  }

  /**
   *  Get a clone of this <code>Device</code>.
   *  @return a shalow copy of this instance.
   */
  public Object clone() {
    return new Device(this);
  }

  /**
   *  Test the equality of this <code>Device</code>
   *  and an object <code>obj</code>.
   *
   *  @param obj the object this test will be performed with
   *  @return false if <code>obj</code> is not of <code>Device</code> class,
   *          true if all attributes are equal.
   */
  public boolean equals(Object obj) {
    if (obj instanceof Device) {
      Device cmp=(Device)obj;
      if (getDeviceStatus()!=cmp.getDeviceStatus() &&
           (getDeviceStatus()==null || !getDeviceStatus().equals(cmp.getDeviceStatus()))
          ) return false;
      if (getHasLocation()!=cmp.getHasLocation() &&
           (getHasLocation()==null || !getHasLocation().equals(cmp.getHasLocation()))
          ) return false;
      if (getInstanceName()!=cmp.getInstanceName() &&
           (getInstanceName()==null || !getInstanceName().equals(cmp.getInstanceName()))
          ) return false;
      if (getTimestamp()!=cmp.getTimestamp() &&
           (getTimestamp()==null || !getTimestamp().equals(cmp.getTimestamp()))
          ) return false;
      return true;
    }
    return false;
  }
}

/*
 * DeviceData.java
 *
 * Generated by Protege plugin Beanynizer.
 * Changes will be lost!
 */
package ontology.onto;



/**
```

```
 *  Java class for concept Device of SmartHouseOntology ontology.
 */
public abstract class DeviceData          extends PhysicalObject implements nuggets.INugget
{
        //-------- constants ----------

        /** Predefined value "ON" for slot deviceStatus. */
        public static String  DEVICESTATUS_ON  = "ON";
        /** Predefined value "OFF" for slot deviceStatus. */
        public static String  DEVICESTATUS_OFF  = "OFF";

        //-------- attributes ----------

        /** Attribute for slot deviceStatus. */
        protected  String  devicestatus;

        /** Attribute for slot timestamp. */
        protected  String  timestamp;

        //-------- constructors --------

        /**
         *  Default Constructor. <br>
         *  Create a new <code>Device</code>.
         */
        public DeviceData()  { //
        }

        //-------- accessor methods --------

        /**
         *  Get the deviceStatus of this Device.
         * @return deviceStatus
         */
        public String  getDeviceStatus() {
                return this.devicestatus;
        }

        /**
         *  Set the deviceStatus of this Device.
         * @param devicestatus the value to be set
         */
        public void  setDeviceStatus(String devicestatus) {
                this.devicestatus = devicestatus;
        }

        /**
         *  Get the timestamp of this Device.
         * @return timestamp
         */
        public String  getTimestamp() {
                return this.timestamp;
        }

        /**
         *  Set the timestamp of this Device.
         * @param timestamp the value to be set
         */
        public void  setTimestamp(String timestamp) {
                this.timestamp = timestamp;
        }

        //-------- object methods --------

        /**
         *  Get a string representation of this Device.
         *  @return The string representation.
```

```
	 */
	public String toString() {
		return "Device("
		+ "devicestatus="+getDeviceStatus()
		+ ", haslocation="+getHasLocation()
		+ ", instancename="+getInstanceName()
		+ ", timestamp="+getTimestamp()
		+ ")";
	}

	//--------- nuggets methods ---------

	/**
	 * Persist this Device using the nuggets utility.
	 * @param c
	 */
	public void _persist(nuggets.ICruncher c) {
		// declare references
		int idHasLocation = c.declare(haslocation);
		// persist the nugget
		c.startConcept(this);
		if (devicestatus!=null)
			c.put("devicestatus", devicestatus);
		if (idHasLocation>0)
			c.put("haslocation", idHasLocation);
		if (instancename!=null)
			c.put("instancename", instancename);
		if (timestamp!=null)
			c.put("timestamp", timestamp);
	}

	/**
	 * Restore this Device
	 * @param a the name of the attribute
	 * @param v the value of the attribute
	 */
	public void _set(String a, Object v) { //
		switch(hash(a)) {
		case 0: haslocation =  (Location)v; return;
		case 1: timestamp =  (String)v; return;
		case 2: devicestatus =  (String)v; return;
		case 3: instancename =  (String)v; return;
		}
	}

	private static final int hash(String name) {
		return ((1411735230*name.charAt(0))>>>15)%4;
	}

}
/*
 * $class.javaName.java
 *
 * Generated by Protege plugin Beanynizer.
 * This class implements the functionality of Human.
 * Feel free to change.
 *
 */
package ontology.onto;


/**
 *  Editable Java class for concept <code>Human</code> of SmartHouseOntology ontology.
 */
public class Human  extends HumanData implements nuggets.INugget
{
	//-------- constructors --------
```

```
/**
 *  Default Constructor. <br>
 *  Create a new <code>Human</code>.
 */
public Human() {
		// Empty constructor required for JavaBeans (do not remove).
}

/**
 *  Init Constructor. <br>
 *  Create a new <code>Human</code>.
 * @param haslocation
 * @param instancename
 */
public Human(Location haslocation, String instancename) {
		// Constructor using required slots (change if desired).
		setHasLocation(haslocation);
		setInstanceName(instancename);
}


/**
 *  Clone Constructor. <br>
 *  Create a new <code>Human</code>.<br>
 *  Copy all attributes from <code>proto</code> to this instance.
 *
 *  @param proto The prototype instance.
 */
public Human(Human proto) {
		setHasLocation(proto.getHasLocation());
		setInstanceName(proto.getInstanceName());
}

//-------- custom code --------


//-------- Object methods -----

/**
 *  Get a string representation of this <code>Human</code>.
 *  @return The string representation.
 */
public String toString() {
		return "Human("
		+ "haslocation="+getHasLocation()
		+ ", instancename="+getInstanceName()
		+ ")";
}

/**
 *  Get a clone of this <code>Human</code>.
 *  @return a shalow copy of this instance.
 */
public Object clone() {
		return new Human(this);
}

/**
 *  Test the equality of this <code>Human</code>
 *  and an object <code>obj</code>.
 *
 *  @param obj the object this test will be performed with
 *  @return false if <code>obj</code> is not of <code>Human</code> class,
 *          true if all attributes are equal.
 */
public boolean equals(Object obj) {
```

```
                if (obj instanceof Human) {
                        Human cmp=(Human)obj;
                        if (getHasLocation()!=cmp.getHasLocation() &&
                                        (getHasLocation()==null
|| !getHasLocation().equals(cmp.getHasLocation()))
                        ) return false;
                        if (getInstanceName()!=cmp.getInstanceName() &&
                                        (getInstanceName()==null
|| !getInstanceName().equals(cmp.getInstanceName()))
                        ) return false;
                        return true;
                }
                return false;
        }
}

/*
 * HumanData.java
 *
 * Generated by Protege plugin Beanynizer.
 * Changes will be lost!
 */
package ontology.onto;


/**
 *  Java class for concept Human of SmartHouseOntology ontology.
 */
public abstract class HumanData        extends Individ implements nuggets.INugget
{
        //-------- constants ----------

        //-------- attributes ----------

        //-------- constructors --------

        /**
         *  Default Constructor. <br>
         *  Create a new <code>Human</code>.
         */
        public HumanData()  { //
        }

        //-------- accessor methods --------

        //-------- object methods --------

        /**
         *  Get a string representation of this Human.
         *  @return The string representation.
         */
        public String toString() {
                return "Human("
                + "haslocation="+getHasLocation()
                + ", instancename="+getInstanceName()
                + ")";
        }

        //--------- nuggets methods ---------

        /**
         * Persist this Human using the nuggets utility.
         * @param c
         */
        public void _persist(nuggets.ICruncher c) {
                // declare references
                int idHasLocation = c.declare(haslocation);
                // persist the nugget
                c.startConcept(this);
                if (idHasLocation>0)
                        c.put("haslocation", idHasLocation);
                if (instancename!=null)
                        c.put("instancename", instancename);
        }

        /**
         * Restore this Human
         * @param a the name of the attribute
         * @param v the value of the attribute
         */
        public void _set(String a, Object v) { //
                switch(hash(a)) {
                case 0: haslocation =  (Location)v; return;
                case 1: instancename =  (String)v; return;
                }
        }

        private static final int hash(String name) {
                return ((1042128340*name.charAt(0))>>>15)%2;
        }

}

/*
 * $class.javaName.java
 *
 * Generated by Protege plugin Beanynizer.
 * This class implements the functionality of Location.
 * Feel free to change.
 *
 */
package ontology.onto;


/**
 *  Editable Java class for concept <code>Location</code> of SmartHouseOntology ontology.
 */
public class Location  extends LocationData implements nuggets.INugget
{
  //-------- constructors --------

  /**
   *  Default Constructor. <br>
   *  Create a new <code>Location</code>.
   */
  public Location() {
    // Empty constructor required for JavaBeans (do not remove).
  }

  /**
   *  Init Constructor. <br>
   *  Create a new <code>Location</code>.
   * @param instancename
   */
  public Location(String instancename) {
    // Constructor using required slots (change if desired).
    setInstanceName(instancename);
  }


  /**
   *  Clone Constructor. <br>
   *  Create a new <code>Location</code>.<br>
   *  Copy all attributes from <code>proto</code> to this instance.
   *
   *  @param proto The prototype instance.
   */
  public Location(Location proto) {
    setInstanceName(proto.getInstanceName());
  }

  //-------- custom code --------


  //-------- Object methods -----

  /**
   *  Get a string representation of this <code>Location</code>.
   *  @return The string representation.
   */
  public String toString() {
    return "Location("
         + "instancename="+getInstanceName()
         + ")";
  }

  /**
   *  Get a clone of this <code>Location</code>.
   *  @return a shalow copy of this instance.
   */
  public Object clone() {
    return new Location(this);
  }

  /**
   *  Test the equality of this <code>Location</code>
   *  and an object <code>obj</code>.
   *
   *  @param obj the object this test will be performed with
   *  @return false if <code>obj</code> is not of <code>Location</code> class,
   *          true if all attributes are equal.
   */
  public boolean equals(Object obj) {
    if (obj instanceof Location) {
      Location cmp=(Location)obj;
      if (getInstanceName()!=cmp.getInstanceName() &&
           (getInstanceName()==null || !getInstanceName().equals(cmp.getInstanceName()))
          ) return false;
      return true;
    }
    return false;
  }
}

/*
 * LocationData.java
 *
 * Generated by Protege plugin Beanynizer.
 * Changes will be lost!
 */
package ontology.onto;

/**
 *  Java class for concept Location of SmartHouseOntology ontology.
 */
public abstract class LocationData         extends ContextEntity implements nuggets.INugget
{
                //-------- constants ----------

                //-------- attributes ----------
	//-------- constructors --------

	/**
	 *  Default Constructor. <br>
	 *  Create a new <code>Location</code>.
	 */
	public LocationData()  { //
	}

	//-------- accessor methods --------

	//-------- object methods --------

	/**
	 *  Get a string representation of this Location.
	 *  @return The string representation.
	 */
	public String toString() {
		return "Location("
		+ "instancename="+getInstanceName()
      + ")";
	}

	//--------- nuggets methods ---------

	/**
	 * Persist this Location using the nuggets utility.
	 * @param c
	 */
	public void _persist(nuggets.ICruncher c) {
		// declare references
		// persist the nugget
		c.startConcept(this);
		if (instancename!=null)
			c.put("instancename", instancename);
	}

	/**
	 * Restore this Location
	 * @param a the name of the attribute
	 * @param v the value of the attribute
	 */
	public void _set(String a, Object v) { //
		instancename =  (String)v;
	}
}

package ontology.onto;

/**
 *  Generated Java class for ontology SmartHouseOntology.
 */
public class SmartHouse
{
	//-------- constants --------

	/** The name of the ontology. */
	public static final String	ONTOLOGY_NAME	= "SmartHouseOntology";


	static	{ // static part
	}
}
```

Jena Model classes
Package ContextModel

File: CreateModel.java
This class contains methods for manipulating an Ontological model*/

```
package contextModel;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Vector;
import org.mindswap.pellet.jena.PelletReasonerFactory;
import com.hp.hpl.jena.ontology.DataRange;
import com.hp.hpl.jena.ontology.Individual;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntProperty;
import com.hp.hpl.jena.ontology.OntResource;
import com.hp.hpl.jena.rdf.model.Literal;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.rdf.model.Resource;

public class CreateModel {

        private String NS;
        private OntModel model;


        /* constructor for memory model; namespace is the ontology namespace, filepath is the path to the ontology*/
        public CreateModel(String namespace,String filepath){
                NS = namespace;
                /*create an ontology model, to which we attach a Pellet Reasoner*/
                model = ModelFactory.createOntologyModel( PelletReasonerFactory.THE_SPEC );
                model.read(filepath);

        }

        /* constructor for persistent model*/

        public CreateModel(OntModel m, String namespace){
                model=m;
                NS=namespace;
        }

        public OntModel getModel(){
                return this.model;
        }

        public String getNamespace(){
                return this.NS;
        }

        /*returns a Vector of subclasses of a certain class*/
        public Vector getSubclasses(String cls){

                OntClass refClass = model.getOntClass(this.NS + cls);
                Vector classArray=new Vector();
                for (Iterator i = refClass.listSubClasses(); i.hasNext(); ) {
                        OntClass subClass = (OntClass) i.next();
                        /*class Nothing should not be included, even if generated by the reasoner */
                        if (!(subClass.getLocalName().equals("Nothing"))){
                                classArray.add(subClass.getLocalName());
                        }
                }
                return classArray;
        }

        /*returns a Vector of direct subclasses of a certain class*/
        public Vector getDirectSubclasses(String cls){

                OntClass refClass = model.getOntClass(this.NS + cls);
                Vector classArray=new Vector();
                for (Iterator i = refClass.listSubClasses(true); i.hasNext(); ) {
                        OntClass subClass = (OntClass) i.next();
                        if (!(subClass.getLocalName().equals("Nothing"))){
                                classArray.add(subClass.getLocalName());
                        }
                }
                return classArray;
        }

        /*returns a Vector of Individuals of a certain class*/
        public Vector getIndividuals(String cls){
                OntClass refClass = model.getOntClass(this.NS + cls);

                Vector classArray=new Vector();
                for (Iterator i = refClass.listInstances(); i.hasNext(); ) {
                        Individual individual = (Individual) i.next();
                        classArray.add(individual.getLocalName());
                }
                return classArray;
        }

        /*returns a List of Properties of a certain class*/
        public ArrayList getProperties(String cls){
                OntClass refclass = model.getOntClass(this.NS + cls);
                ArrayList<OntProperty> classArray = new ArrayList<OntProperty>();

                for (Iterator i = refclass.listDeclaredProperties(); i.hasNext(); )
                {
                        OntProperty property=(OntProperty) i.next();
                        classArray.add(property);
                }
                return classArray;
        }

        /*returns a Individual instance from an instanceName*/
        public Individual getIndividualFromURI(String localName){
                return model.getIndividual(this.NS+localName);
        }

        /*returns a Property instance from an instanceName*/
        public Property getPropertyFromURI(String localName){
                return model.getProperty(this.NS+localName);
        }

        /*returns a a property value for an Individual given by its instanceName*/
        public String getIndividualPropertyValue(String ind, String prop){
                Individual i =getIndividualFromURI(ind);
                String uri=this.NS+prop;
                Property p=this.model.getProperty(uri);
                Literal lVal=null;
                RDFNode val = i.getPropertyValue(p );
                if (val instanceof Literal) {
                        lVal = (Literal) val;
                        return (lVal.getString());
                }
                else {
                        OntResource rVal = (OntResource) val.as( OntResource.class );
                        return(rVal.getLocalName());
                }
```

```
	}

	/*returns a Vector of allowable values for a property*/
	public Vector getAllowableValues(String prop) {
		String uri=this.NS+prop;
		OntProperty p=this.model.getOntProperty(uri);
		Vector valArray=new Vector();
		OntResource r = p.getRange();
		if (r != null) {
			if (r.isDataRange()) {
				DataRange dr = r.asDataRange();
				for (Iterator itr = dr.listOneOf(); itr.hasNext();) {
					Literal l = (Literal) itr.next();
					valArray.add(l.getString());

				}
			} else {
			}
		}
		return valArray;
	}

	/*returns an Individual instance, given its name and class*/
	public Individual createIndividual(String individualName, String individualClass) {
		Resource res = model.getResource(this.NS+individualClass);
		Individual ind=model.createIndividual(this.NS + individualName,res);
		return ind;
	}

	/*adds a property and a property value to an individual*/
	public void addPropertyValue(String propertyLiteral, String propertyName, String individualName) {
		Individual ind = model.getIndividual(this.NS+individualName);
		Property prop = model.getProperty(this.NS+propertyName);
		ind.addProperty(prop,propertyLiteral);
		}

	/*sets a property and the property's value to an individual*/
	/*isLiteral indicates weather the property is a Literal or Resource*/
	public void setPropertyValue(String individualName, String property,String propertyValue, boolean isLiteral) {
		Individual ind = model.getIndividual(this.NS+ individualName);
		Property prop = model.getProperty(this.NS+property);
		if (isLiteral){
			Literal l =model.createTypedLiteral(propertyValue);
			ind.setPropertyValue(prop,l);
		}
		else{
			Resource sensorLocation = model.getResource(this.NS+propertyValue);
			ind.setPropertyValue(prop,sensorLocation);
		}
	}
}
```

/* */
File: Connection.java
This class contains methods for creating and manipulating a database connection, for Jena persistent models */

```
package com.rfc.tools.jena.manager.domain;

import com.hp.hpl.jena.db.DBConnection;
import com.hp.hpl.jena.db.IDBConnection;

public class Connection
{
    private String name = "";
    private String driver = "";
    private String url = "";
    private String user = "";
    private String password = "";
    private String lastResult = "";

    private IDBConnection conn;

    public Connection(String name, String driver, String url, String user, String pasword)
    {
        this.name = name;
        this.driver = driver;
        this.url = url;
        this.user = user;
        this.password = pasword;
    }

    public Connection()
    {
    }

    public void connect()
    {
        try
        {
            if (isConnected())
            {
                disconnect();
            }

            Class.forName(driver);
            conn = new DBConnection(url, user, password, "MySQL");
            testConnection();
            lastResult = "Connected to " + name;
        }
        catch (Throwable e)
        {
            lastResult = e.toString();
            e.printStackTrace();
        }
    }

    public void disconnect()
    {
        if (conn != null)
        {
            try
            {
                conn.close();
                lastResult = "Disconnected from " + name;
            }
            catch (Throwable e)
            {
```

```
            e.printStackTrace();
        }
    }
    conn = null;
}

public boolean isConnected()
{
    return conn != null;
}

private void testConnection()
{
    conn.getAllModelNames();
}

IDBConnection getJenaConnection()
{
    return conn;
}

@Override
public String toString()
{
    return name;
}

public String getName()
{
    return name;
}

public String getDriver()
{
    return driver;
}

public String getUrl()
{
    return url;
}

public String getUser()
{
    return user;
}

public String getPassword()
{
    return password;
}

public String getLastResult()
{
    return lastResult;
}}
```

File: CreateModel.java
This class contains methods forquery execution over a jena model*/

```
package com.rfc.tools.jena.manager.domain;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.ModelMaker;

public class QueryExecutor
{
  private OntModel model;

  public void addModel(Database database, AppModel subModel)
  {
    ModelMaker maker = ModelFactory.createModelRDBMaker(database.getConnection()
        .getJenaConnection());
    Model newModel = maker.getModel(subModel.getName());
    if (model == null)
    {
      model = ModelFactory.createOntologyModel(OntModelSpec.RDFS_MEM, newModel);
    }
    else
    {
      model.addSubModel(newModel);
    }
  }

  public ResultSet execute(String query)
  {
    ResultSet results = null;

    try
    {
      QueryExecution execution = QueryExecutionFactory.create(query, model);

      results = execution.execSelect();

      execution.close();
    }
    catch (Throwable e)
    {
      e.printStackTrace();
    }

    return results;
  }
}
```

Package: PresentationLayer
File: MonitoringGUI.agent.xml
- ADF file for the MonitorinGUI agent

```
<agent xmlns="http://jadex.sourceforge.net/jadex"
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance

        xsi:schemaLocation="http://jadex.sourceforge.net/jadex
                http://jadex.sourceforge.net/jadex-0.96.xsd"
        name="MonitoringGUI"
        package="PresentationLayer">

        <imports>
                <import>java.util.logging.*</import>
                <import>jadex.adapter.fipa.*</import>
                <import>jadex.util.*</import>
                <import>java.util.*</import>
                <import>jadex.runtime.*</import>
                <import>jadex.runtime.impl.*</import>
                <import>jadex.planlib.*</import>
                <import>ontology.lowerOntology.*</import>
        </imports>

        <beliefs>
          <belief name="sensorType" exported="true" class="Vector">
            <fact>new Vector()</fact>
   </belief>
        </beliefs>

        <plans>
                <!-- Initial plan for creating and updating the gui. -->
                <plan name="monitoringGui">
                        <body class="StartMonitoringGUIPlan"/>
                        </plan>
                <plan name="robotsGui">
                        <body class="StartGUIPlan"/>
                </plan>

                <plan name="GuiDataPlan">
                        <body class="GuiDataPlan"/>
                </plan>
                <!--<plan name="GuiDataPlan2">
                        <body class="GuiDataPlan2"/>
                        <trigger>
                                <messageevent ref="context_info"/>
                        </trigger>
                </plan>
                -->
        </plans>

        <events>
                <messageevent name="requestContextData" direction="send" type="fipa">
                        <parameter name="performative" class="String" direction="fixed">
                                <value>SFipa.REQUEST</value>
                        </parameter>
                        <parameter name="conversation-id" class="String" direction="fixed">
                        <value>"contextSharingStart"</value>
        </parameter>
      </messageevent>

                <messageevent name="context_info" direction="receive" type="fipa">
      <parameter name="performative" class="String" direction="fixed">
        <value>SFipa.INFORM</value>
      </parameter>
      <parameter name="conversation-id" class="String" direction="fixed">
```

```
            <value>"contextSharingStart"</value>
         </parameter>

         <parameter name="language" class="String" direction="fixed">
            <value>SFipa.NUGGETS_XML</value>
          </parameter>
          <parameter  name="content-class" class="Class" direction="fixed">
            <value>java.util.Vector.class</value>
          </parameter>
          <parameter name="ontology" class="String" direction="fixed">
            <value>ontology.lowerOntology.SmartHouse.ONTOLOGY_NAME</value>
          </parameter>
       </messageevent>

                        <messageevent name="message" direction="send" type="fipa">
                                    <parameter name="performative" class="String" direction="fixed">
                                                <value>SFipa.INFORM</value>
                                    </parameter>
                                    <parameter name="in-reply-to" class="String" direction="fixed">
            <value>"requestContextData"</value>
         </parameter>
                                    <parameter name="language" class="String" direction="fixed">
                                     <value>SFipa.NUGGETS_XML</value>
                                    </parameter>
                                    <parameter name="ontology" class="String" direction="fixed">
                        <value>ontology.lowerOntology.SmartHouse.ONTOLOGY_NAME</value>
                        </parameter>
                        </messageevent>
            </events>

            <properties>
                        <!-- Only log outputs >= level are printed. -->
    <property name="logging.level">Level.INFO</property>
                        <!-- The default parent handler prints out log messages on the console. -->
    <property name="logging.useParentHandlers">true</property>
  </properties>

            <configurations>
                        <configuration name="default">

                                    <plans>
                                                <initialplan ref="robotsGui"/>
                                                <initialplan ref="GuiDataPlan"/>
                                                <initialplan ref="monitoringGui"/>
                                                </plans>
                        </configuration>
            </configurations>

</agent>
```

Package: PresentationLayer
File: GuiDataPlan.java
- Plan implementation for context information retrieval

```
package PresentationLayer;
import java.util.Vector;
import ontology.lowerOntology.Sensor;
import jadex.adapter.fipa.AgentIdentifier;
import jadex.model.IMBelief;
import jadex.model.IMBeliefbase;
import jadex.runtime.IExternalAccess;
import jadex.runtime.IMessageEvent;
import jadex.runtime.Plan;

public class GuiDataPlan extends Plan {
        private IExternalAccess agent;
        private static final int TIMEOUT = 15000;

        public GuiDataPlan() {

        }

        public void body()
        {
                IMBeliefbase model;
                IMBelief belief;

                getLogger().info("Sensor information");
                jadex.adapter.fipa.AgentIdentifier receiver;
                receiver = new AgentIdentifier("KBAgent",true);
                IMessageEvent me = createMessageEvent("requestContextData");
                me.setContent("getSensorTypes");

                me.getParameterSet(jadex.adapter.fipa.SFipa.RECEIVERS).addValue(receiver);

                IMessageEvent reply = sendMessageAndWait(me);
                getLogger().info("Sending message");
                Vector sensorTypes=new Vector();
                sensorTypes=(Vector)reply.getContent();
                getLogger().info("reply from KB AGENT"+sensorTypes.elementAt(0));

          getBeliefbase().getBelief("sensorType").setFact(sensorTypes);

          for(int i=0;i<sensorTypes.size();i++){
                  model = (IMBeliefbase)getBeliefbase().getModelElement();
                  belief = model.createBelief((String)sensorTypes.elementAt(i), Vector.class, -1, "false");
                  this.getBeliefbase().registerBelief(belief);
                }

          IMessageEvent me1= createMessageEvent("requestContextData");
                me1.setContent("getSensors");
                me1.getParameterSet(jadex.adapter.fipa.SFipa.RECEIVERS).addValue(receiver);
                IMessageEvent reply1 = sendMessageAndWait(me1);
                Vector sensors=new Vector();
                sensors=(Vector)reply1.getContent();

                Vector v=new Vector();
                for(int i=0; i<sensors.size();i++){
                        v=(Vector)sensors.elementAt(i);
                        this.getBeliefbase().getBelief((String)sensorTypes.elementAt(i)).setFact(v);
                        for(int j=0; j<v.size();j++){
                                Sensor s=(Sensor)v.elementAt(j);


                        }
                }
```

```
                        getLogger().info("reply from KB AGENT SENSORS");

            }
}
```

Package: PresentationLayer
File: SensorGui.java
- Plan implementation for the monitoring and update interface

```
/*
 * Created by JFormDesigner on Thu May 15 02:10:38 EEST 2008
 */

package PresentationLayer.util;
import com.jgoodies.uif_lite.panel.*;
import jadex.adapter.fipa.AgentIdentifier;
import jadex.runtime.IExternalAccess;
import jadex.runtime.IMessageEvent;
import jadex.runtime.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;

import javax.swing.*;
import javax.swing.border.*;

import ontology.lowerOntology.Device;
import ontology.lowerOntology.Location;
import ontology.lowerOntology.Sensor;
import util.AgentMessage;
import KBLayer.CreateModel;

import com.hp.hpl.jena.ontology.Individual;
import com.jgoodies.forms.factories.*;
import com.jgoodies.forms.layout.*;



/**
 * @author Hoszu Amalia
 */
public class SensorGui extends JPanel implements ActionListener {
            private CreateModel model;
            private Vector sensors;
            private Vector devices;
            private Vector states;
            //private Vector sensorType;
            private String uri;
            private IExternalAccess agent;
            private static final int TIMEOUT = 15000;

            public SensorGui(final IExternalAccess agent) {

               this.agent=agent;
                        model=new CreateModel("http://www.owl-
ontologies.com/Ontology1207603095.owl#","file:ontology/sensor.owl");
                        sensors=new Vector();
                        devices=new Vector();
                        states=new Vector( model.getIndividuals("State"));
                        initComponents();

            }
```

```
public JPanel getPanel(){
		return this.panel2;
}

private void button2ActionPerformed(ActionEvent e) {
		// TODO add your code here
}





private void initComponents() {
		// JFormDesigner - Component initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
		// Generated using JFormDesigner Evaluation license - Hoszu Amalia
		action1 = new AddSensor();
		action2 = new AddNewDevice();
		action3 = new EnableSensor();
		action4 = new SetSensorData();
		action5 = new DisableSensor();
		action6 = new EnableDevice();
		action7 = new DisableDevice();
		action8 = new SetDeviceData();
		panel2 = new JPanel();
		panel5 = new JPanel();
		tabbedPane2 = new JTabbedPane();
		panel4 = new JPanel();
		label6 = new JLabel();
		comboBox7 = new JComboBox();
		label7 = new JLabel();
		comboBox8 = new JComboBox();
		label8 = new JLabel();
		comboBox9 = new JComboBox();
		label9 = new JLabel();
		comboBox10 = new JComboBox();
		button5 = new JButton();
		label16 = new JLabel();
		comboBox15 = new JComboBox();
		button1 = new JButton();
		label10 = new JLabel();
		textField1 = new JTextField();
		button2 = new JButton();
		simpleInternalFrame1 = new SimpleInternalFrame();
		panel7 = new JPanel();
		comboBox11 = new JComboBox();
		label12 = new JLabel();
		comboBox12 = new JComboBox();
		label13 = new JLabel();
		label14 = new JLabel();
		comboBox13 = new JComboBox();
		button10 = new JButton();
		comboBox14 = new JComboBox();
		button11 = new JButton();
		label15 = new JLabel();
		textField3 = new JTextField();
		button4 = new JButton();
		simpleInternalFrame2 = new SimpleInternalFrame();
		label11 = new JLabel();
		panel6 = new JPanel();
		label1 = new JLabel();
		comboBox16 = new JComboBox();
		label27 = new JLabel();
		textField2 = new JTextField();
		label28 = new JLabel();
		comboBox17 = new JComboBox();
		label17 = new JLabel();
		comboBox18 = new JComboBox();
```

```
label18 = new JLabel();
comboBox19 = new JComboBox();
button6 = new JButton();
simpleInternalFrame3 = new SimpleInternalFrame();
panel8 = new JPanel();
label2 = new JLabel();
comboBox20 = new JComboBox();
label3 = new JLabel();
comboBox22 = new JComboBox();
label29 = new JLabel();
textField4 = new JTextField();
label30 = new JLabel();
comboBox21 = new JComboBox();
label20 = new JLabel();
comboBox23 = new JComboBox();
button7 = new JButton();
simpleInternalFrame4 = new SimpleInternalFrame();
panel1 = new JPanel();
label32 = new JLabel();
label33 = new JLabel();
label4 = new JLabel();
label5 = new JLabel();
label19 = new JLabel();
comboBox1 = new JComboBox();
button3 = new JButton();
label24 = new JLabel();
comboBox4 = new JComboBox();
comboBox6 = new JComboBox();
comboBox27 = new JComboBox();
comboBox26 = new JComboBox();
label21 = new JLabel();
comboBox2 = new JComboBox();
button8 = new JButton();
label25 = new JLabel();
comboBox24 = new JComboBox();
comboBox28 = new JComboBox();
label22 = new JLabel();
comboBox3 = new JComboBox();
label23 = new JLabel();
comboBox5 = new JComboBox();
button12 = new JButton();
button9 = new JButton();
label26 = new JLabel();
comboBox25 = new JComboBox();
comboBox29 = new JComboBox();
label31 = new JLabel();
comboBox30 = new JComboBox();
simpleInternalFrame5 = new SimpleInternalFrame();
panel3 = new JPanel();
scrollPane1 = new JScrollPane();
textArea1 = new JTextArea();
action9 = new StateSelection();
action10 = new setMaryState();
action12 = new setJohnState();
action13 = new setJLittleJohnState();
action14 = new setLittleJaneState();
CellConstraints cc = new CellConstraints();

//======== panel2 ========
{
        panel2.setBackground(Color.white);

        // JFormDesigner evaluation mark
        panel2.setBorder(new javax.swing.border.CompoundBorder(
                new javax.swing.border.TitledBorder(new javax.swing.border.EmptyBorder(0, 0, 0,
0),
                        "JFormDesigner Evaluation", javax.swing.border.TitledBorder.CENTER,
                                                javax.swing.border.TitledBorder.BOTTOM, new java.awt.Font("Dialog",
java.awt.Font.BOLD, 12),
                                                java.awt.Color.red), panel2.getBorder()));
panel2.addPropertyChangeListener(new java.beans.PropertyChangeListener(){public void
propertyChange(java.beans.PropertyChangeEvent e){if("border".equals(e.getPropertyName()))throw new
RuntimeException();}});


                              panel2.setLayout(new GridBagLayout());
                              ((GridBagLayout)panel2.getLayout()).columnWidths = new int[] {558, 0};
                              ((GridBagLayout)panel2.getLayout()).rowHeights = new int[] {0, 0, 314, 0};
                              ((GridBagLayout)panel2.getLayout()).columnWeights = new double[] {1.0, 1.0E-4};
                              ((GridBagLayout)panel2.getLayout()).rowWeights = new double[] {0.0, 0.0, 1.0, 1.0E-4};

                              //======== panel5 ========
                              {
                                        panel5.setBackground(new Color(205, 225, 255));
                                        panel5.setLayout(new FormLayout(
                                                "default:grow",
                                                "default, default"));

                                        //======== tabbedPane2 ========
                                        {
                                                tabbedPane2.setBackground(Color.white);
                                                tabbedPane2.setBorder(new
SoftBevelBorder(SoftBevelBorder.LOWERED));

                                                //======== panel4 ========
                                                {
                                                        panel4.setBackground(Color.white);
                                                        panel4.setLayout(new FormLayout(
                                                                "47px, left:5px, 87px, left:5px, 105px, left:5px, 53px,
left:5px, 77px, left:5px, pref:grow",
                                                                "60px, top:5px, 32px, top:5px, 32px, top:5px, 32px,
top:5px, 32px, top:5px, 32px, top:5px, 32px, top:5px, pref:grow"));

                                                        //---- label6 ----
                                                        label6.setText("Sensor Type");
                                                        panel4.add(label6, cc.xywh(3, 3, 9, 1));

                                                        //---- comboBox7 ----
                                                        Vector sensorType=new Vector();
                                                        sensorType=model.getSubclasses("Sensor");

          //sensorType=(Vector)agent.getBeliefbase().getBelief("sensorType").getFact();

          comboBox7 = new JComboBox(sensorType);

          comboBox7.setSelectedIndex(0);

          comboBox7.addActionListener(this);
                                                        panel4.add(comboBox7, cc.xy(5, 3));

                                                        //---- label7 ----
                                                        label7.setText("Sensor ID");
                                                        panel4.add(label7, cc.xy(3, 5));

                                                        //---- comboBox8 ----
                                                        comboBox8.addActionListener(this);

          comboBox8.addItemListener(new ItemListener() {

          public void itemStateChanged(ItemEvent e) {

          if (e.getStateChange() == ItemEvent.SELECTED) {

                    setCurrentSensorLocation(comboBox8,comboBox9);
```

```
            setCurrentSensorValue(comboBox8,comboBox10);

            setCurrentSensorStatus(comboBox8,comboBox15);

            setCurrentSensorTimestamp(comboBox8,textField1);

                                                                                }



}

});
                                        panel4.add(comboBox8, cc.xy(5, 5));

                                        //---- label8 ----
                                        label8.setText("Sensor Location");
                                        panel4.add(label8, cc.xy(3, 7));

                                        //---- comboBox9 ----
                                        Vector sensorLoc=new Vector();
                                        sensorLoc=model.getIndividuals("IndoorSpace");
                                        comboBox9 = new JComboBox(sensorLoc);
                                        comboBox9.addActionListener(this);
                                        panel4.add(comboBox9, cc.xy(5, 7));

                                        //---- label9 ----
                                        label9.setText("Sensor Value");
                                        panel4.add(label9, cc.xy(3, 9));

                                        //---- comboBox10 ----
                                        Vector sensorValue=new Vector();

sensorValue=model.getAllowableValues("sensorValue");

comboBox10 = new JComboBox(sensorValue);

//comboBox10.setSelectedItem(s.getSensorValue());

comboBox10.addActionListener(this);
                                        panel4.add(comboBox10, cc.xy(5, 9));

                                        //---- button5 ----
                                        button5.setAction(action3);
                                        panel4.add(button5, cc.xy(9, 9));

                                        //---- label16 ----
                                        label16.setText("Sensor Status");
                                        panel4.add(label16, cc.xy(3, 11));

                                        //---- comboBox15 ----
                                        Vector sensorStatus=new Vector();
                                        sensorStatus=model.getAllowableValues("deviceStatus");
                                        comboBox15 = new JComboBox(sensorStatus);
                                        comboBox15.addActionListener(this);
                                        panel4.add(comboBox15, cc.xy(5, 11));

                                        //---- button1 ----
                                        button1.setAction(action5);
                                        panel4.add(button1, cc.xy(9, 11));

                                        //---- label10 ----
                                        label10.setText("Timestamp");
                                        panel4.add(label10, cc.xy(3, 13));
```

```
						//---- textField1 ----
						textField1.setEnabled(false);
						textField1.setEnabled(false);
						panel4.add(textField1, cc.xy(5, 13));

						//---- button2 ----
						button2.setAction(action4);
						button2.addActionListener(new ActionListener() {
							public void actionPerformed(ActionEvent e) {
								button2ActionPerformed(e);
							}
						});
						panel4.add(button2, cc.xy(9, 13));

						//======== simpleInternalFrame1 ========
						{
							simpleInternalFrame1.setTitle("text");
							Container simpleInternalFrame1ContentPane =
simpleInternalFrame1.getContentPane();
							simpleInternalFrame1ContentPane.setLayout(new
FormLayout(
								"default",
								"default"));
						}
						panel4.add(simpleInternalFrame1, cc.xywh(1, 1, 11, 15));
					}
					tabbedPane2.addTab("Sensors", new
ImageIcon(getClass().getResource("/images/sensor2.png")), panel4);


					//======== panel7 ========
					{
						panel7.setBackground(Color.white);
						panel7.setLayout(new FormLayout(
							new ColumnSpec[] {
								new ColumnSpec(Sizes.dluX(31)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(Sizes.dluX(58)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(Sizes.dluX(63)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(Sizes.dluX(35)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(Sizes.dluX(46)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(ColumnSpec.FILL,
Sizes.DEFAULT, FormSpec.DEFAULT_GROW)
							},
							new RowSpec[] {
								new RowSpec(Sizes.dluY(40)),
								FormFactory.LINE_GAP_ROWSPEC,
								new RowSpec(Sizes.dluY(21)),
								FormFactory.LINE_GAP_ROWSPEC,
								new RowSpec(Sizes.dluY(21)),
								FormFactory.LINE_GAP_ROWSPEC,
								new RowSpec(Sizes.dluY(21)),
								FormFactory.LINE_GAP_ROWSPEC,
								new RowSpec(Sizes.dluY(21)),
								FormFactory.LINE_GAP_ROWSPEC,
								new RowSpec(Sizes.dluY(21)),
								FormFactory.LINE_GAP_ROWSPEC,
                                        new RowSpec(RowSpec.CENTER,
Sizes.dluY(63), FormSpec.DEFAULT_GROW),
                                        FormFactory.LINE_GAP_ROWSPEC,
                                        FormFactory.DEFAULT_ROWSPEC,
                                        FormFactory.LINE_GAP_ROWSPEC,
                                        FormFactory.DEFAULT_ROWSPEC
                                }));

                        //---- comboBox11 ----
                        Vector deviceType=new Vector();
                        deviceType=model.getDirectSubclasses("Device");
                        comboBox11 = new JComboBox(deviceType);
                        comboBox11.setSelectedIndex(0);
                        comboBox11.addActionListener(this);

                        panel7.add(comboBox11, cc.xy(5, 3));

                        //---- label12 ----
                        label12.setText("Device ID");
                        panel7.add(label12, cc.xy(3, 5));

                        //---- comboBox12 ----
                        comboBox12.addActionListener(this);
                        comboBox12.addItemListener(new ItemListener() {
                        public void itemStateChanged(ItemEvent e) {
                                if (e.getStateChange() == ItemEvent.SELECTED) {

    setCurrentDeviceLocation(comboBox12,comboBox13);

    setCurrentDeviceValue(comboBox12,comboBox14);

    setCurrentDeviceTimestamp(comboBox12,textField3);
                                }
                        }
                        });
                        panel7.add(comboBox12, cc.xy(5, 5));

                        //---- label13 ----
                        label13.setText("Device Location");
                        panel7.add(label13, cc.xy(3, 7));

                        //---- label14 ----
                        label14.setText("Device Status");
                        panel7.add(label14, cc.xy(3, 9));

                        //---- comboBox13 ----
                        Vector deviceLoc=new Vector();
                        deviceLoc=model.getIndividuals("IndoorSpace");
                        comboBox13 = new JComboBox(deviceLoc);
                        comboBox13.addActionListener(this);
                        panel7.add(comboBox13, cc.xy(5, 7));

                        //---- button10 ----
                        button10.setSelectedIcon(null);
                        button10.setAction(action6);
                        panel7.add(button10, cc.xy(9, 7));

                        //---- comboBox14 ----
                        Vector deviceValue=new Vector();
                        deviceValue=model.getAllowableValues("deviceStatus");
                        comboBox14 = new JComboBox(deviceValue);
                        comboBox14.addActionListener(this);
                        panel7.add(comboBox14, cc.xy(5, 9));

                        //---- button11 ----
                        button11.setAction(action7);
                        panel7.add(button11, cc.xy(9, 9));
```

```
					//---- label15 ----
					label15.setText("Timestamp");
					panel7.add(label15, cc.xy(3, 11));

					//---- textField3 ----
					textField3.setEnabled(false);
					textField3.setEnabled(false);
					panel7.add(textField3, cc.xy(5, 11));

					//---- button4 ----
					button4.setAction(action8);
					button4.addActionListener(new ActionListener() {
						public void actionPerformed(ActionEvent e) {
							button2ActionPerformed(e);
						}
					});
					panel7.add(button4, cc.xy(9, 11));

					//======== simpleInternalFrame2 ========
					{
						simpleInternalFrame2.setTitle("text");
						Container simpleInternalFrame2ContentPane =
simpleInternalFrame2.getContentPane();
						simpleInternalFrame2ContentPane.setLayout(new
FormLayout(
							"default",
							"default"));
					}
					panel7.add(simpleInternalFrame2, cc.xywh(1, 1, 11, 13));

					//---- label11 ----
					label11.setText("Device Type");
					panel7.add(label11, cc.xywh(3, 3, 9, 1));
				}
				tabbedPane2.addTab("Devices", new
ImageIcon(getClass().getResource("/images/device1.png")), panel7);


				//======== panel6 ========
				{
					panel6.setBackground(Color.white);
					panel6.setLayout(new FormLayout(
						new ColumnSpec[] {
							new ColumnSpec(Sizes.dluX(31)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
							new ColumnSpec(Sizes.dluX(58)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
							new ColumnSpec(Sizes.dluX(63)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
							new ColumnSpec(Sizes.dluX(35)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
							new ColumnSpec(Sizes.dluX(46)),
							new ColumnSpec(ColumnSpec.FILL,
Sizes.DEFAULT, FormSpec.DEFAULT_GROW),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
							new ColumnSpec(ColumnSpec.FILL,
Sizes.DEFAULT, FormSpec.DEFAULT_GROW)
						},
						new RowSpec[] {
```

```
                                                                                new RowSpec(Sizes.dluY(40)),
                                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                                new RowSpec(Sizes.dluY(21)),
                                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                                new RowSpec(Sizes.dluY(21)),
                                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                                new RowSpec(Sizes.dluY(21)),
                                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                                new RowSpec(Sizes.dluY(21)),
                                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                                new RowSpec(Sizes.dluY(21)),
                                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                                new RowSpec(RowSpec.CENTER,
Sizes.DEFAULT, FormSpec.DEFAULT_GROW),

                                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                                new RowSpec(Sizes.DLUY11)
                                                                        }));

                                                                //---- label1 ----
                                                                label1.setText("Sensor Type");
                                                                panel6.add(label1, cc.xy(3, 3));

                                                                //---- comboBox16 ----
                                                                Vector sensorType=new Vector();

        sensorType=model.getSubclasses("Sensor");

        comboBox16 = new JComboBox(sensorType);

        comboBox16.setSelectedIndex(0);

        comboBox16.addActionListener(this);
                                                                panel6.add(comboBox16, cc.xy(5, 3));

                                                                //---- label27 ----
                                                                label27.setText("Sensor ID");
                                                                panel6.add(label27, cc.xy(3, 5));
                                                                panel6.add(textField2, cc.xy(5, 5));

                                                                //---- label28 ----
                                                                label28.setText("Sensor Location");
                                                                panel6.add(label28, cc.xy(3, 7));

                                                                //---- comboBox17 ----
                                                                Vector sensorLoc=new Vector();
                                                                sensorLoc=model.getIndividuals("IndoorSpace");
                                                                comboBox17 = new JComboBox(sensorLoc);
                                                                comboBox17.addActionListener(this);
                                                                panel6.add(comboBox17, cc.xy(5, 7));

                                                                //---- label17 ----
                                                                label17.setText("Sensor Value");
                                                                panel6.add(label17, cc.xy(3, 9));

                                                                //---- comboBox18 ----
                                                                Vector sensorValue=new Vector();

        sensorValue=model.getAllowableValues("sensorValue");

        comboBox18 = new JComboBox(sensorValue);

        //comboBox10.setSelectedItem(s.getSensorValue());

        comboBox18.addActionListener(this);
                                                                panel6.add(comboBox18, cc.xy(5, 9));

                                                                //---- label18 ----
```

```
                                                label18.setText("Sensor Status");
                                                panel6.add(label18, cc.xy(3, 11));

                                                //---- comboBox19 ----
                                                Vector sensorStatus=new Vector();
                                                sensorStatus=model.getAllowableValues("deviceStatus");
                                                comboBox19 = new JComboBox(sensorStatus);
                                                comboBox19.addActionListener(this);
                                                panel6.add(comboBox19, cc.xy(5, 11));

                                                //---- button6 ----
                                                button6.setAction(action1);
                                                panel6.add(button6, cc.xy(9, 11));

                                                //======== simpleInternalFrame3 ========
                                                {
                                                        simpleInternalFrame3.setTitle("text");
                                                        Container simpleInternalFrame3ContentPane =
simpleInternalFrame3.getContentPane();
                                                        simpleInternalFrame3ContentPane.setLayout(new
FormLayout(
                                                                "default",
                                                                "default"));
                                                }
                                                panel6.add(simpleInternalFrame3, cc.xywh(1, 1, 13, 13));
                                        }
                                        tabbedPane2.addTab("Sensor Management", new
ImageIcon(getClass().getResource("/images/sensor1.png")), panel6);


                                        //======== panel8 ========
                                        {
                                                panel8.setBackground(Color.white);
                                                panel8.setLayout(new FormLayout(
                                                        new ColumnSpec[] {
                                                                new ColumnSpec(Sizes.dluX(29)),

        FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
                                                                new ColumnSpec(Sizes.dluX(58)),

        FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
                                                                new ColumnSpec(Sizes.dluX(63)),

        FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
                                                                new ColumnSpec(Sizes.dluX(35)),

        FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
                                                                new ColumnSpec(Sizes.dluX(47)),
                                                                new ColumnSpec(ColumnSpec.FILL,
Sizes.DEFAULT, FormSpec.DEFAULT_GROW),

        FormFactory.LABEL_COMPONENT_GAP_COLSPEC,

        FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
                                                                new ColumnSpec(ColumnSpec.FILL,
Sizes.DEFAULT, FormSpec.DEFAULT_GROW)
                                                        },
                                                        new RowSpec[] {
                                                                new RowSpec(Sizes.dluY(40)),
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                new RowSpec(Sizes.dluY(21)),
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                new RowSpec(Sizes.dluY(21)),
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                new RowSpec(Sizes.dluY(20)),
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                new RowSpec(Sizes.dluY(21)),
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                new RowSpec(Sizes.dluY(21)),
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                new RowSpec(RowSpec.CENTER,
Sizes.DEFAULT, FormSpec.DEFAULT_GROW),
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                new RowSpec(Sizes.DLUY11),
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                FormFactory.DEFAULT_ROWSPEC,
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                FormFactory.DEFAULT_ROWSPEC,
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                FormFactory.DEFAULT_ROWSPEC,
                                                                FormFactory.LINE_GAP_ROWSPEC,
                                                                FormFactory.DEFAULT_ROWSPEC
                                                        }));

                                                //---- label2 ----
                                                label2.setText("Device Type");
                                                panel8.add(label2, cc.xy(3, 3));

                                                //---- comboBox20 ----
                                                Vector sensorType=new Vector();

        sensorType=model.getDirectSubclasses("Device");

        comboBox20 = new JComboBox(sensorType);

        comboBox20.setSelectedIndex(0);

        comboBox20.addActionListener(this);
                                                panel8.add(comboBox20, cc.xy(5, 3));

                                                //---- label3 ----
                                                label3.setText("Device");
                                                panel8.add(label3, cc.xy(3, 5));

                                                //---- comboBox22 ----
                                                Vector deviceSubType=new Vector();

        deviceSubType=model.getDirectSubclasses(comboBox20.getSelectedItem().toString());;
                                                comboBox22 = new JComboBox(deviceSubType);
                                                comboBox22.setSelectedIndex(0);
                                                comboBox22.addActionListener(this);
                                                panel8.add(comboBox22, cc.xy(5, 5));

                                                //---- label29 ----
                                                label29.setText("Device ID");
                                                panel8.add(label29, cc.xy(3, 7));
                                                panel8.add(textField4, cc.xy(5, 7));

                                                //---- label30 ----
                                                label30.setText("Device Location");
                                                panel8.add(label30, cc.xy(3, 9));

                                                //---- comboBox21 ----
                                                Vector sensorLoc=new Vector();
                                                sensorLoc=model.getIndividuals("IndoorSpace");
                                                comboBox21 = new JComboBox(sensorLoc);
                                                comboBox21.addActionListener(this);
                                                panel8.add(comboBox21, cc.xy(5, 9));

                                                //---- label20 ----
                                                label20.setText("Device Status");
                                                panel8.add(label20, cc.xy(3, 11));

                                                //---- comboBox23 ----
```

```
						Vector sensorStatus=new Vector();
						sensorStatus=model.getAllowableValues("deviceStatus");
						comboBox23 = new JComboBox(sensorStatus);
						comboBox23.addActionListener(this);
						panel8.add(comboBox23, cc.xy(5, 11));

						//---- button7 ----
						button7.setAction(action2);
						panel8.add(button7, cc.xy(9, 11));

						//======== simpleInternalFrame4 ========
						{
simpleInternalFrame4.setTitle("text");
Container simpleInternalFrame4ContentPane = simpleInternalFrame4.getContentPane();
simpleInternalFrame4ContentPane.setLayout(new FormLayout("default", "default"));
						}
panel8.add(simpleInternalFrame4, cc.xywh(1, 1, 13, 23));
					}
tabbedPane2.addTab("Device Management", new ImageIcon(getClass().getResource("/images/device2.png")), panel8);


					//======== panel1 ========
					{
						panel1.setLayout(new FormLayout(
							new ColumnSpec[] {
								new ColumnSpec(Sizes.dluX(29)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(Sizes.dluX(59)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(Sizes.dluX(71)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								FormFactory.DEFAULT_COLSPEC,

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(Sizes.dluX(75)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(Sizes.dluX(63)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(Sizes.dluX(31)),

	FormFactory.LABEL_COMPONENT_GAP_COLSPEC,
								new ColumnSpec(Sizes.dluX(68))
							},
							new RowSpec[] {
								FormFactory.DEFAULT_ROWSPEC,
								FormFactory.LINE_GAP_ROWSPEC,
								new RowSpec(Sizes.dluY(13)),
								FormFactory.LINE_GAP_ROWSPEC,
								new RowSpec(Sizes.dluY(69)),
								FormFactory.LINE_GAP_ROWSPEC,
								new RowSpec(Sizes.dluY(69)),
								FormFactory.LINE_GAP_ROWSPEC,
								new RowSpec(Sizes.dluY(69)),
								FormFactory.LINE_GAP_ROWSPEC,
								new RowSpec(Sizes.dluY(69)),
								FormFactory.LINE_GAP_ROWSPEC,
								FormFactory.DEFAULT_ROWSPEC
							}));

						//---- label32 ----
						label32.setText("Entity");
						panel1.add(label32, cc.xy(3, 3));
```

```
                                        //---- label33 ----
                                        label33.setText("State");
                                        panel1.add(label33, cc.xy(5, 3));

                                        //---- label4 ----
                                        label4.setText("Location");
                                        panel1.add(label4, cc.xy(11, 3));

                                        //---- label5 ----
                                        label5.setText("Activity");
                                        panel1.add(label5, cc.xy(15, 3));

                                        //---- label19 ----
                                        label19.setText("Mary");
                                        panel1.add(label19, cc.xy(3, 5));

                                        //---- comboBox1 ----
                                        comboBox1.setAction(action9);

comboBox1=ComboBoxRenderer.initCombo(comboBox1,states,label24);

comboBox1.setSelectedItem(model.getIndividualPropertyValue("Mary","hasState"));
                                        comboBox1.addActionListener(this);
                                        panel1.add(comboBox1, cc.xy(5, 5));

                                        //---- button3 ----
                                        button3.setAction(action10);
                                        panel1.add(button3, cc.xy(7, 5));
                                        panel1.add(label24, cc.xy(9, 5));

                                        //---- comboBox4 ----
                                        Vector sensorLoc=new Vector();
                                        sensorLoc=model.getIndividuals("Location");
                                        comboBox4 = new JComboBox(sensorLoc);

comboBox4.setSelectedItem(model.getIndividualPropertyValue("Mary","hasLocation"));
                                        panel1.add(comboBox4, cc.xy(11, 5));

                                        //---- comboBox6 ----
                                        Vector activity=new Vector();
                                        activity=model.getIndividuals("Activity");
                                        comboBox6 = new JComboBox(activity);

comboBox6.setSelectedItem(model.getIndividualPropertyValue("Mary","hasActivity"));
                                        panel1.add(comboBox6, cc.xy(15, 5));

                                        //---- comboBox27 ----
                                        sensorLoc=new Vector();
                                        sensorLoc=model.getIndividuals("Location");
                                        comboBox27 = new JComboBox(sensorLoc);

comboBox27.setSelectedItem(model.getIndividualPropertyValue("LittleJohn","hasLocation"));
                                        panel1.add(comboBox27, cc.xy(11, 11));

                                        //---- comboBox26 ----
                                        sensorLoc=new Vector();
                                        sensorLoc=model.getIndividuals("Location");
                                        comboBox26 = new JComboBox(sensorLoc);

comboBox26.setSelectedItem(model.getIndividualPropertyValue("Jane","hasLocation"));
                                        panel1.add(comboBox26, cc.xy(11, 9));

                                        //---- label21 ----
                                        label21.setText("John");
                                        panel1.add(label21, cc.xy(3, 7));
```

```
                                //---- comboBox2 ----
                                comboBox2.setAction(null);

comboBox2=ComboBoxRenderer.initCombo(comboBox2,states,label25);
                                comboBox2.addActionListener(this);

comboBox2.setSelectedItem(model.getIndividualPropertyValue("John","hasState"));
                                panel1.add(comboBox2, cc.xy(5, 7));

                                //---- button8 ----
                                button8.setAction(action12);
                                panel1.add(button8, cc.xy(7, 7));
                                panel1.add(label25, cc.xy(9, 7));

                                //---- comboBox24 ----
                                sensorLoc=new Vector();
                                sensorLoc=model.getIndividuals("Location");
                                comboBox24 = new JComboBox(sensorLoc);

comboBox24.setSelectedItem(model.getIndividualPropertyValue("John","hasLocation"));
                                panel1.add(comboBox24, cc.xy(11, 7));

                                //---- comboBox28 ----
                                activity=new Vector();
                                activity=model.getIndividuals("Activity");
                                comboBox28 = new JComboBox(activity);

comboBox28.setSelectedItem(model.getIndividualPropertyValue("John","hasActivity"));
                                panel1.add(comboBox28, cc.xy(15, 7));

                                //---- label22 ----
                                label22.setText("Jane");
                                panel1.add(label22, cc.xy(3, 9));

                                //---- comboBox3 ----

comboBox3=ComboBoxRenderer.initCombo(comboBox3,states,label26);
                                comboBox3.addActionListener(this);

comboBox3.setSelectedItem(model.getIndividualPropertyValue("Jane","hasState"));
                                panel1.add(comboBox3, cc.xy(5, 9));

                                //---- label23 ----
                                label23.setText("LittleJohn");
                                panel1.add(label23, cc.xy(3, 11));

                                //---- comboBox5 ----

comboBox5=ComboBoxRenderer.initCombo(comboBox5,states,label31);
                                comboBox5.addActionListener(this);

comboBox5.setSelectedItem(model.getIndividualPropertyValue("LittleJohn","hasState"));

                                panel1.add(comboBox5, cc.xy(5, 11));

                                //---- button12 ----
                                button12.setAction(action13);
                                panel1.add(button12, cc.xy(7, 11));

                                //---- button9 ----
                                button9.setAction(action14);
                                panel1.add(button9, cc.xy(7, 9));
                                panel1.add(label26, cc.xy(9, 9));
                                panel1.add(comboBox25, cc.xy(11, 9));

                                //---- comboBox29 ----
                                activity=new Vector();
```

```
                                                        activity=model.getIndividuals("Activity");
                                                        comboBox29 = new JComboBox(activity);

        comboBox29.setSelectedItem(model.getIndividualPropertyValue("Jane","hasActivity"));
                                                        panel1.add(comboBox29, cc.xy(15, 9));
                                                        panel1.add(label31, cc.xy(9, 11));

                                                        //---- comboBox30 ----
                                                        activity=new Vector();
                                                        activity=model.getIndividuals("Activity");
                                                        comboBox30 = new JComboBox(activity);

        comboBox30.setSelectedItem(model.getIndividualPropertyValue("LittleJohn","hasActivity"));
                                                        panel1.add(comboBox30, cc.xy(15, 11));

                                                        //======== simpleInternalFrame5 ========
                                                        {
                                                                simpleInternalFrame5.setTitle("text");
                                                                Container simpleInternalFrame5ContentPane =
simpleInternalFrame5.getContentPane();
                                                                simpleInternalFrame5ContentPane.setLayout(new
FormLayout(
                                                                        "default",
                                                                        "default"));
                                                        }
                                                        panel1.add(simpleInternalFrame5, cc.xywh(1, 1, 15, 13));
                                                }
                                                tabbedPane2.addTab("State Detector", new
ImageIcon(getClass().getResource("/images/emoticon.png")), panel1);

                                        }
                                        panel5.add(tabbedPane2, cc.xy(1, 1));
                                }
                                panel2.add(panel5, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0,
                                        GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                                        new Insets(0, 0, 5, 0), 0, 0));

                                //======== panel3 ========
                                {
                                        panel3.setBorder(new CompoundBorder(
                                                new TitledBorder("Action Log"),
                                                new EmptyBorder(5, 5, 5, 5)));
                                        panel3.setBackground(new Color(205, 225, 255));
                                        panel3.setLayout(new GridBagLayout());
                                        ((GridBagLayout)panel3.getLayout()).columnWidths = new int[] {95, 0};
                                        ((GridBagLayout)panel3.getLayout()).rowHeights = new int[] {243, 0};
                                        ((GridBagLayout)panel3.getLayout()).columnWeights = new double[] {1.0, 1.0E-4};
                                        ((GridBagLayout)panel3.getLayout()).rowWeights = new double[] {1.0, 1.0E-4};
                                        setCurrentComboValues();

                                        //======== scrollPane1 ========
                                        {
                                                scrollPane1.setViewportView(textArea1);
                                        }
                                        panel3.add(scrollPane1, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0,
                                                GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                                                new Insets(0, 0, 0, 0), 0, 0));
                                }
                                panel2.add(panel3, new GridBagConstraints(0, 1, 1, 2, 0.0, 0.0,
                                        GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                                        new Insets(0, 0, 0, 0), 0, 0));
                        }
                        // JFormDesigner - End of component initialization  //GEN-END:initComponents
        }

        // JFormDesigner - Variables declaration - DO NOT MODIFY  //GEN-BEGIN:variables
        // Generated using JFormDesigner Evaluation license - Hoszu Amalia
```

```
private AddSensor action1;
private AddNewDevice action2;
private EnableSensor action3;
private SetSensorData action4;
private DisableSensor action5;
private EnableDevice action6;
private DisableDevice action7;
private SetDeviceData action8;
private JPanel panel2;
private JPanel panel5;
private JTabbedPane tabbedPane2;
private JPanel panel4;
private JLabel label6;
private JComboBox comboBox7;
private JLabel label7;
private JComboBox comboBox8;
private JLabel label8;
private JComboBox comboBox9;
private JLabel label9;
private JComboBox comboBox10;
private JButton button5;
private JLabel label16;
private JComboBox comboBox15;
private JButton button1;
private JLabel label10;
private JTextField textField1;
private JButton button2;
private SimpleInternalFrame simpleInternalFrame1;
private JPanel panel7;
private JComboBox comboBox11;
private JLabel label12;
private JComboBox comboBox12;
private JLabel label13;
private JLabel label14;
private JComboBox comboBox13;
private JButton button10;
private JComboBox comboBox14;
private JButton button11;
private JLabel label15;
private JTextField textField3;
private JButton button4;
private SimpleInternalFrame simpleInternalFrame2;
private JLabel label11;
private JPanel panel6;
private JLabel label1;
private JComboBox comboBox16;
private JLabel label27;
private JTextField textField2;
private JLabel label28;
private JComboBox comboBox17;
private JLabel label17;
private JComboBox comboBox18;
private JLabel label18;
private JComboBox comboBox19;
private JButton button6;
private SimpleInternalFrame simpleInternalFrame3;
private JPanel panel8;
private JLabel label2;
private JComboBox comboBox20;
private JLabel label3;
private JComboBox comboBox22;
private JLabel label29;
private JTextField textField4;
private JLabel label30;
private JComboBox comboBox21;
private JLabel label20;
private JComboBox comboBox23;
```

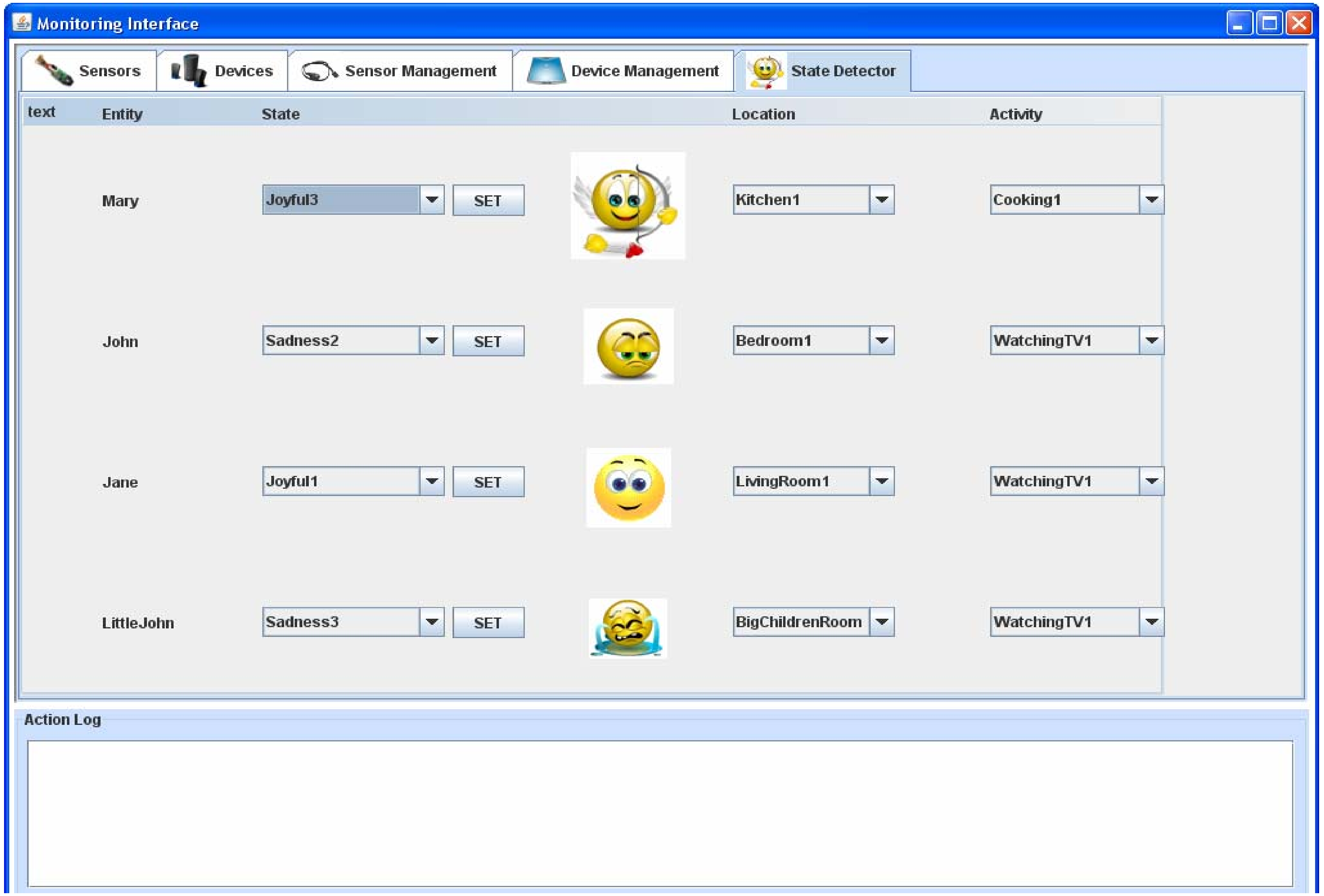
```
	private JButton button7;
	private SimpleInternalFrame simpleInternalFrame4;
	private JPanel panel1;
	private JLabel label32;
	private JLabel label33;
	private JLabel label4;
	private JLabel label5;
	private JLabel label19;
	private JComboBox comboBox1;
	private JButton button3;
	private JLabel label24;
	private JComboBox comboBox4;
	private JComboBox comboBox6;
	private JComboBox comboBox27;
	private JComboBox comboBox26;
	private JLabel label21;
	private JComboBox comboBox2;
	private JButton button8;
	private JLabel label25;
	private JComboBox comboBox24;
	private JComboBox comboBox28;
	private JLabel label22;
	private JComboBox comboBox3;
	private JLabel label23;
	private JComboBox comboBox5;
	private JButton button12;
	private JButton button9;
	private JLabel label26;
	private JComboBox comboBox25;
	private JComboBox comboBox29;
	private JLabel label31;
	private JComboBox comboBox30;
	private SimpleInternalFrame simpleInternalFrame5;
	private JPanel panel3;
	private JScrollPane scrollPane1;
	private JTextArea textArea1;
	private StateSelection action9;
	private setMaryState action10;
	private setJohnState action12;
	private setJLittleJohnState action13;
	private setLittleJaneState action14;
	// JFormDesigner - End of variables declaration  //GEN-END:variables

	public void actionPerformed(ActionEvent e) {
			if (e.getSource() == comboBox7) {
					setCurrentSensorComboValues();
			}
			else if (e.getSource() == comboBox11) {
					setCurrentDeviceComboValues();
			}
			else if (e.getSource() == comboBox20) {
					Vector deviceSubType=new Vector();
					deviceSubType=model.getDirectSubclasses(comboBox20.getSelectedItem().toString());;
					comboBox22.removeAllItems();
					for (int i=0;i<deviceSubType.size(); i++) {
							String s=(String)deviceSubType.elementAt(i);
							comboBox22.addItem(s);
					}
					comboBox22.setSelectedIndex(0);


			}
			else if (e.getSource() == comboBox1) {
			String sel = (String)comboBox1.getSelectedItem();
 ComboBoxRenderer.updateLabel(sel,label24);
}
```

```
		else if (e.getSource() == comboBox2) {
			String sel = (String)comboBox2.getSelectedItem();
	    ComboBoxRenderer.updateLabel(sel,label25);
		}
		else if (e.getSource() == comboBox3) {
			String sel = (String)comboBox3.getSelectedItem();
	    ComboBoxRenderer.updateLabel(sel,label26);
	  }
		else if (e.getSource() == comboBox5) {
			String sel = (String)comboBox5.getSelectedItem();
	    ComboBoxRenderer.updateLabel(sel,label31);
	   }

	}

	public void createSensor(String individualClass,String sensorValue, String timestamp, String deviceStatus,String
instanceName,String location) {
		Individual x=model.createIndividual(instanceName,individualClass);
		model.setPropertyValue(instanceName, "sensorValue", sensorValue,true);
		model.setPropertyValue(instanceName, "timestamp",timestamp,true);
		model.setPropertyValue(instanceName, "instanceName", instanceName,true);
		model.setPropertyValue(instanceName, "deviceStatus", deviceStatus,true);
		model.setPropertyValue(instanceName,"hasLocation",location,false);

	}

	public void createDevice(String individualClass,String timestamp, String deviceStatus,String instanceName,String
location) {
		Individual x=model.createIndividual(instanceName,individualClass);
		model.setPropertyValue(instanceName, "timestamp",timestamp,true);
		model.setPropertyValue(instanceName, "instanceName", instanceName,true);
		model.setPropertyValue(instanceName, "deviceStatus", deviceStatus,true);
		model.setPropertyValue(instanceName,"hasLocation",location,false);

	}



	public Vector getGuiContent(String sensorType) {
		Vector sensorInd=new Vector();
		sensorInd=model.getIndividuals(sensorType);
		for (int i=0;i<sensorInd.size(); i++) {
			String crt=(String)sensorInd.elementAt(i);
			Location loc=new Location(model.getIndividualPropertyValue(crt,"hasLocation"));
			Sensor s=new Sensor(model.getIndividualPropertyValue(crt,"deviceStatus"),
								loc,

	model.getIndividualPropertyValue(crt,"instanceName"),

	model.getIndividualPropertyValue(crt,"sensorValue"),

	model.getIndividualPropertyValue(crt,"timestamp")
								);

			sensors.addElement(s);
		}
		return sensors;
	}

	public Vector getGuiContent2(String deviceType) {
		Vector sensorInd=new Vector();
		sensorInd=model.getIndividuals(deviceType);
		for (int i=0;i<sensorInd.size(); i++) {
			String crt=(String)sensorInd.elementAt(i);
			Location loc=new Location(model.getIndividualPropertyValue(crt,"hasLocation"));
			Device s=new Device(model.getIndividualPropertyValue(crt,"deviceStatus"),
								loc,
model.getIndividualPropertyValue(crt,"instanceName"),

model.getIndividualPropertyValue(crt,"timestamp")
                                                                                                    );

                        devices.addElement(s);
            }
            return devices;
}


public Sensor getCurrentSensor(String sensorId) {
            Sensor s=new Sensor();
            for (int i=0;i<sensors.size(); i++) {
                        s=(Sensor)sensors.elementAt(i);
                        if (s.getInstanceName().equals(sensorId)){
                                    return s;
                        }
            }
            return s;
}

public Device getCurrentDevice(String deviceId) {
            Device s=new Device();
            for (int i=0;i<devices.size(); i++) {
                        s=(Device)devices.elementAt(i);
                        if (s.getInstanceName().equals(deviceId)){
                                    return s;
                        }
            }
            return s;
}


public void setCurrentComboValues() {
            setCurrentSensorComboValues() ;
            setCurrentDeviceComboValues() ;

}


public void setCurrentSensorComboValues() {
            setCurrentSensorId(comboBox7,comboBox8);
            setCurrentSensorLocation(comboBox8,comboBox9);
            setCurrentSensorValue(comboBox8,comboBox10);
            setCurrentSensorStatus(comboBox8,comboBox15);
            setCurrentSensorTimestamp(comboBox8,textField1);
}

public void setCurrentSensorIdComboValues() {
            setCurrentSensorLocation(comboBox8,comboBox9);
            setCurrentSensorValue(comboBox8,comboBox10);
            setCurrentSensorStatus(comboBox8,comboBox15);
            setCurrentSensorTimestamp(comboBox8,textField1);
}

public void setCurrentDeviceIdComboValues() {
            setCurrentDeviceLocation(comboBox12,comboBox13);
            setCurrentDeviceValue(comboBox12,comboBox14);
            setCurrentDeviceTimestamp(comboBox12,textField3);
}
public void setCurrentDeviceComboValues() {
            setCurrentDeviceId(comboBox11,comboBox12);
            setCurrentDeviceLocation(comboBox12,comboBox13);
            setCurrentDeviceValue(comboBox12,comboBox14);
            setCurrentDeviceTimestamp(comboBox12,textField3);
```

```
}
public void setCurrentSensorLocation(JComboBox modified,JComboBox update){
		Sensor s= new Sensor();
		s=getCurrentSensor(modified.getSelectedItem().toString());
		update.setSelectedItem(s.getHasLocation().getInstanceName());

}

public void setCurrentSensorValue(JComboBox modified,JComboBox update){
		Sensor s= new Sensor();
		s=getCurrentSensor(modified.getSelectedItem().toString());
		update.setSelectedItem(s.getSensorValue());
}

public void setCurrentSensorStatus(JComboBox modified,JComboBox update){
		Sensor s= new Sensor();
		s=getCurrentSensor(modified.getSelectedItem().toString());
		update.setSelectedItem(s.getDeviceStatus());
}

public void setCurrentDeviceLocation(JComboBox modified,JComboBox update){
		Device s= new Device();
		s=getCurrentDevice(modified.getSelectedItem().toString());
		update.setSelectedItem(s.getHasLocation().getInstanceName());
}

public void setCurrentDeviceValue(JComboBox modified,JComboBox update){
		Device s= new Device();
		s=getCurrentDevice(modified.getSelectedItem().toString());
		update.setSelectedItem(s.getDeviceStatus());
}

public void setCurrentSensorId(JComboBox modified,JComboBox update){
		if (sensors.capacity()!=0)
				sensors.removeAllElements();
		update.removeAllItems();
		sensors=getGuiContent(modified.getSelectedItem().toString());
		Sensor s= new Sensor();
		for (int i=0;i<sensors.size(); i++) {
				s=(Sensor)sensors.elementAt(i);
				update.addItem(s.getInstanceName());
		}
		update.setSelectedIndex(0);

}

public void setCurrentDeviceId(JComboBox modified,JComboBox update){
		if (devices.capacity()!=0)
				devices.removeAllElements();
		update.removeAllItems();
		devices=getGuiContent2(modified.getSelectedItem().toString());
		Device s= new Device();
		for (int i=0;i<devices.size(); i++) {
				s=(Device)devices.elementAt(i);
				update.addItem(s.getInstanceName());
		}
		update.setSelectedIndex(0);
}


public void setCurrentSensorTimestamp(JComboBox modified,JTextField update){
		Sensor s= new Sensor();
		s=getCurrentSensor(modified.getSelectedItem().toString());
		update.setText(s.getTimestamp());
}

public void setCurrentDeviceTimestamp(JComboBox modified,JTextField update){
		Device s= new Device();
		s=getCurrentDevice(modified.getSelectedItem().toString());
		update.setText(s.getTimestamp());
	}



	private class AddSensor extends AbstractAction {
		private AddSensor() {
			// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
			// Generated using JFormDesigner Evaluation license - Hoszu Amalia
			putValue(NAME, "SET");
			// JFormDesigner - End of action initialization  //GEN-END:initComponents
		}

		public void actionPerformed(ActionEvent e) {
			// TODO add your code here
			String individualName=textField2.getText();
			String individualClass=comboBox16.getSelectedItem().toString();
			Individual x=model.createIndividual(individualName,individualClass);
			createSensor(individualClass,
					comboBox18.getSelectedItem().toString(),
					"124564646",
					comboBox19.getSelectedItem().toString(),
					individualName,
					comboBox17.getSelectedItem().toString());

			textArea1.setText(textArea1.getText()+" \n"+ " Added New Sensor: "+individualName
				+"\n\t Sensor Value: "+ comboBox10.getSelectedItem().toString()
					+"\n \t Sensor Value: " + comboBox18.getSelectedItem().toString()
					+"\n \t Sensor Status: " + comboBox19.getSelectedItem().toString()
					+"\n \t Sensor Location: "+comboBox17.getSelectedItem().toString()
					+"\n \t at ");

		}

	}

	private class AddNewDevice extends AbstractAction {
		private AddNewDevice() {
			// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
			// Generated using JFormDesigner Evaluation license - Hoszu Amalia
			putValue(NAME, "SET");
			// JFormDesigner - End of action initialization  //GEN-END:initComponents
		}

		public void actionPerformed(ActionEvent e) {
			// TODO add your code here
			String individualName=textField4.getText();
			String individualClass=comboBox20.getSelectedItem().toString();
			Individual x=model.createIndividual(individualName,individualClass);
			createDevice(individualClass,
					"124564646",
					comboBox23.getSelectedItem().toString(),
					individualName,
					comboBox21.getSelectedItem().toString());
			textArea1.setText(textArea1.getText()+" \n"+ " Added New Device: "+individualName
					+"\n \t Device Status: " + comboBox23.getSelectedItem().toString()
					+"\n \t Device Location: "+comboBox21.getSelectedItem().toString()
					+"\n \t at ");

		}

	}

	private class EnableSensor extends AbstractAction {
		private EnableSensor() {
				// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
				// Generated using JFormDesigner Evaluation license - Hoszu Amalia
				putValue(NAME, "Enable");
				// JFormDesigner - End of action initialization  //GEN-END:initComponents
			}

			public void actionPerformed(ActionEvent e) {
				// TODO add your code here
				String instanceName=comboBox12.getSelectedItem().toString();
				model.setPropertyValue(instanceName, "deviceStatus","ON",true);
				comboBox15.setSelectedItem("ON");
				textArea1.setText(textArea1.getText()+" \n"+ " Sensor instance: "+instanceName+" enabled at
"+textField1.getText());
				setCurrentSensorId(comboBox7,comboBox8);


				}
		}

		private class SetSensorData extends AbstractAction {
			private SetSensorData() {
				// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
				// Generated using JFormDesigner Evaluation license - Hoszu Amalia
				putValue(NAME, "SET");
				// JFormDesigner - End of action initialization  //GEN-END:initComponents
			}

			public void actionPerformed(ActionEvent e) {
				// TODO add your code here
				String instanceName=comboBox8.getSelectedItem().toString();
				model.setPropertyValue(instanceName, "sensorValue",
comboBox10.getSelectedItem().toString(),true);
				model.setPropertyValue(instanceName, "timestamp","54645564561",true);
				model.setPropertyValue(instanceName, "deviceStatus",
comboBox15.getSelectedItem().toString(),true);

		model.setPropertyValue(instanceName,"hasLocation",comboBox9.getSelectedItem().toString(),false);
				textArea1.setText(textArea1.getText()+" \n"+ " Set Sensor: "+instanceName
						+"\n\t Sensor Value: "+ comboBox10.getSelectedItem().toString()
						+"\n \t Sensor Status: " + comboBox15.getSelectedItem().toString()
						+"\n \t Sensor Location: "+comboBox9.getSelectedItem().toString()
						+"\n \t at "+textField1.getText());
				setCurrentSensorId(comboBox7,comboBox8);


				Sensor sensor=new Sensor(comboBox15.getSelectedItem().toString(),
								new
Location(comboBox9.getSelectedItem().toString()),
								instanceName,
								comboBox9.getSelectedItem().toString(),
								((Long)System.currentTimeMillis()).toString()
				);

				jadex.adapter.fipa.AgentIdentifier receiver;
				receiver = new AgentIdentifier("LightSensor",true);
				IMessageEvent me = agent.createMessageEvent("message");
				me.getParameterSet(jadex.adapter.fipa.SFipa.RECEIVERS).addValue(receiver);
				me.setContent(sensor);
				IMessageEvent reply = agent.sendMessageAndWait(me, TIMEOUT);



			}
		}

		private class DisableSensor extends AbstractAction {
			private DisableSensor() {
				// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
				// Generated using JFormDesigner Evaluation license - Hoszu Amalia
				putValue(NAME, "Disable");
				// JFormDesigner - End of action initialization  //GEN-END:initComponents
			}

			public void actionPerformed(ActionEvent e) {
				// TODO add your code here
				String instanceName=comboBox8.getSelectedItem().toString();
				model.setPropertyValue(instanceName, "deviceStatus","OFF",true);
				comboBox15.setSelectedItem("OFF");
				textArea1.setText(textArea1.getText()+" \n"+ " Sensor instance: "+instanceName+" disabled at
"+textField1.getText());
				setCurrentSensorId(comboBox7,comboBox8);


			}
		}

		private class EnableDevice extends AbstractAction {
			private EnableDevice() {
				// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
				// Generated using JFormDesigner Evaluation license - Hoszu Amalia
				putValue(NAME, "Enable");
				// JFormDesigner - End of action initialization  //GEN-END:initComponents
			}

			public void actionPerformed(ActionEvent e) {
				// TODO add your code here
				String instanceName=comboBox12.getSelectedItem().toString();
				model.setPropertyValue(instanceName, "deviceStatus","ON",true);
				comboBox14.setSelectedItem("ON");
				textArea1.setText(textArea1.getText()+" \n"+ " Device instance: "+instanceName+" enabled at
"+textField3.getText());
				setCurrentDeviceId(comboBox11,comboBox12);



			}
		}

		private class DisableDevice extends AbstractAction {
			private DisableDevice() {
				// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
				// Generated using JFormDesigner Evaluation license - Hoszu Amalia
				putValue(NAME, "Disable");
				// JFormDesigner - End of action initialization  //GEN-END:initComponents
			}

			public void actionPerformed(ActionEvent e) {
				// TODO add your code here
				String instanceName=comboBox12.getSelectedItem().toString();
				model.setPropertyValue(instanceName, "deviceStatus","OFF",true);
				comboBox14.setSelectedItem("OFF");
				textArea1.setText(textArea1.getText()+" \n"+ " Device instance: "+instanceName+" disabled at
"+textField3.getText());
				setCurrentDeviceId(comboBox11,comboBox12);

			}
		}

		private class SetDeviceData extends AbstractAction {
			private SetDeviceData() {
				// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
				// Generated using JFormDesigner Evaluation license - Hoszu Amalia
				putValue(NAME, "SET");
				// JFormDesigner - End of action initialization  //GEN-END:initComponents
			}

			public void actionPerformed(ActionEvent e) {
				// TODO add your code here
				String instanceName=comboBox12.getSelectedItem().toString();
				model.setPropertyValue(instanceName, "timestamp","54645564561",true);
				model.setPropertyValue(instanceName, "deviceStatus",
comboBox14.getSelectedItem().toString(),true);

		model.setPropertyValue(instanceName,"hasLocation",comboBox13.getSelectedItem().toString(),false);
				textArea1.setText(textArea1.getText()+" \n"+ " Set Sensor: "+instanceName
						+"\n \t Sensor Status: " + comboBox14.getSelectedItem().toString()
						+"\n \t Sensor Location: "+comboBox13.getSelectedItem().toString()
						+"\n \t at "+textField1.getText());
				setCurrentDeviceId(comboBox11,comboBox12);

			}
		}

		private class StateSelection extends AbstractAction {
			private StateSelection() {
				// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
				// Generated using JFormDesigner Evaluation license - Hoszu Amalia
				putValue(NAME, "selectState");
				// JFormDesigner - End of action initialization  //GEN-END:initComponents
			}

			public void actionPerformed(ActionEvent e) {
				// TODO add your code here
				   String sel = (String)comboBox1.getSelectedItem();
			      ComboBoxRenderer.updateLabel(sel,label24);
			}
		}

		private class setMaryState extends AbstractAction {
			private setMaryState() {
				// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
				// Generated using JFormDesigner Evaluation license - Hoszu Amalia
				putValue(NAME, "SET");
				// JFormDesigner - End of action initialization  //GEN-END:initComponents

			}

			public void actionPerformed(ActionEvent e) {
				// TODO add your code here
				String instanceName="Mary";
				model.setPropertyValue(instanceName, "hasState",
comboBox1.getSelectedItem().toString(),true);

			}
		}

		private class setJohnState extends AbstractAction {
			private setJohnState() {
				// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
				// Generated using JFormDesigner Evaluation license - Hoszu Amalia
				putValue(NAME, "SET");
				// JFormDesigner - End of action initialization  //GEN-END:initComponents
			}

			public void actionPerformed(ActionEvent e) {
				// TODO add your code here
				String instanceName="John";
				model.setPropertyValue(instanceName, "hasState",
comboBox2.getSelectedItem().toString(),true);

			}
```

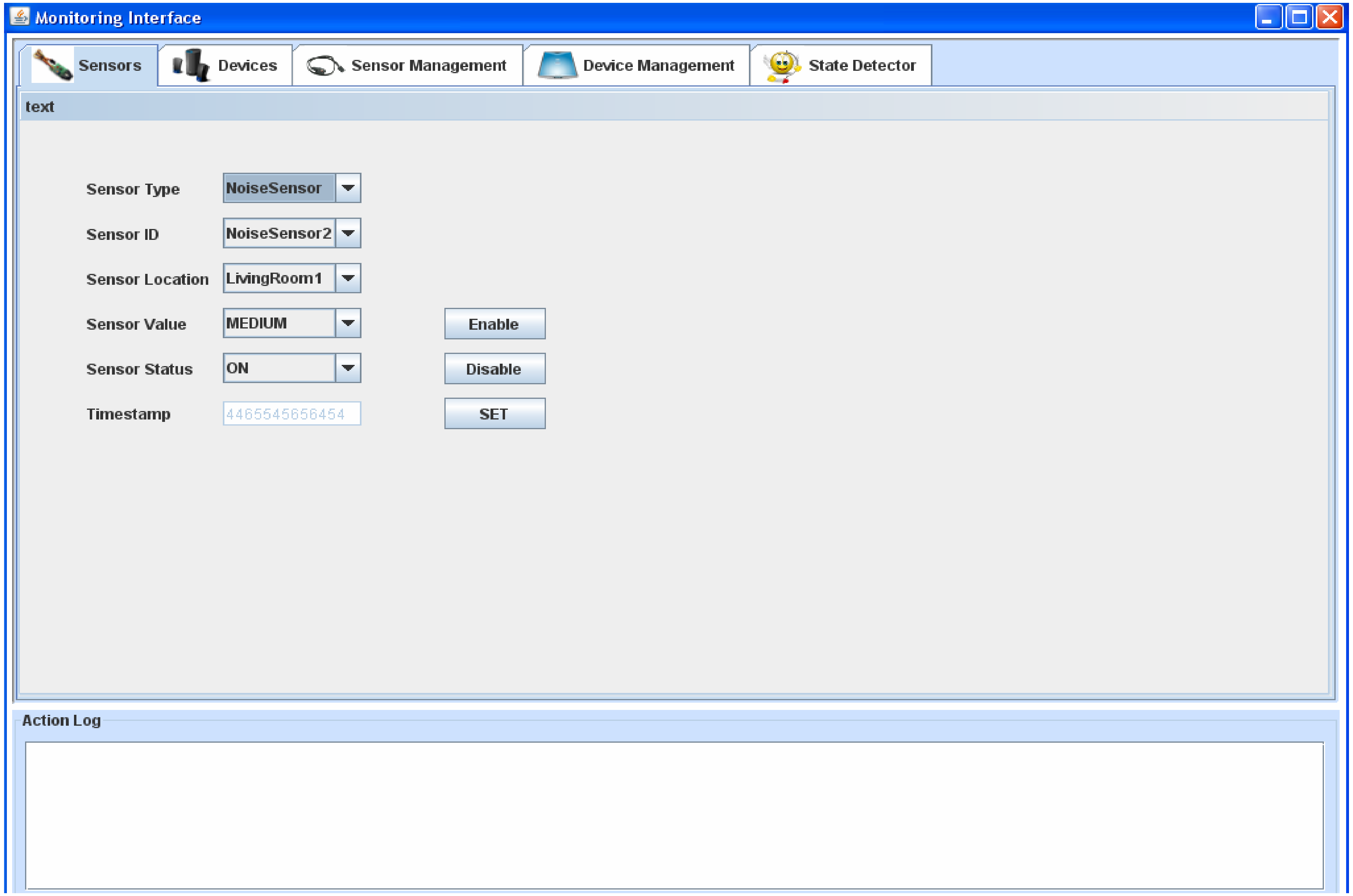
```
	}

	private class setJLittleJohnState extends AbstractAction {
		private setJLittleJohnState() {
			// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
			// Generated using JFormDesigner Evaluation license - Hoszu Amalia
			putValue(NAME, "SET");
			// JFormDesigner - End of action initialization  //GEN-END:initComponents
		}

		public void actionPerformed(ActionEvent e) {
			// TODO add your code here
			String instanceName="LittleJohn";
			model.setPropertyValue(instanceName, "hasState",
comboBox5.getSelectedItem().toString(),true);

		}
	}

	private class setLittleJaneState extends AbstractAction {
		private setLittleJaneState() {
			// JFormDesigner - Action initialization - DO NOT MODIFY  //GEN-BEGIN:initComponents
			// Generated using JFormDesigner Evaluation license - Hoszu Amalia
			putValue(NAME, "SET");
			// JFormDesigner - End of action initialization  //GEN-END:initComponents
		}

		public void actionPerformed(ActionEvent e) {
			// TODO add your code here
			String instanceName="Jane";
			model.setPropertyValue(instanceName, "hasState",
comboBox3.getSelectedItem().toString(),true);

		}
	}
}
```

Monitoring GUI (State Detector)



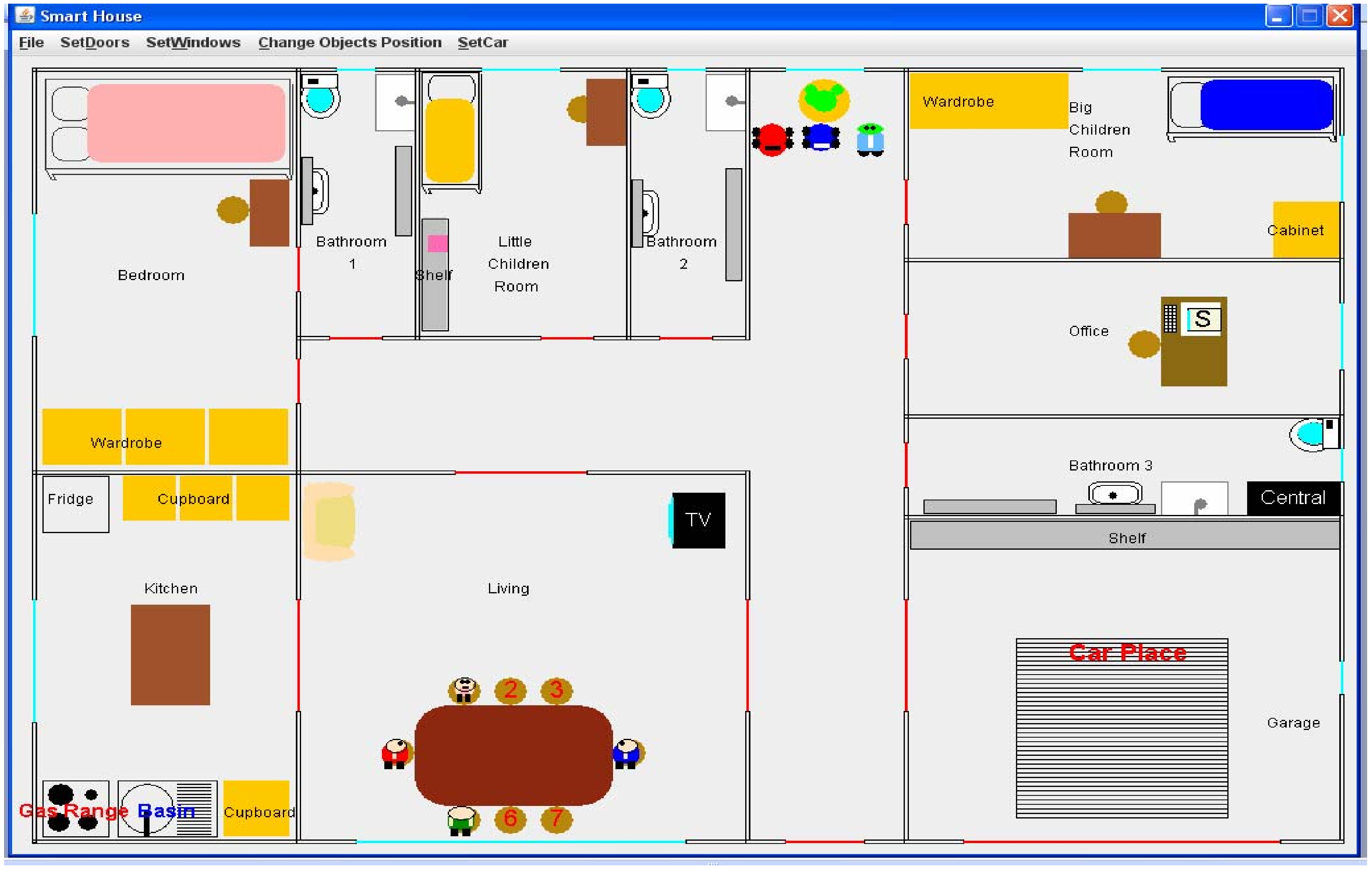
text	Entity	State			Location	Activity
	Mary	Joyful3	SET		Kitchen1	Cooking1
	John	Sadness2	SET		Bedroom1	WatchingTV1
	Jane	Joyful1	SET		LivingRoom1	WatchingTV1
	LittleJohn	Sadness3	SET		BigChildrenRoom	WatchingTV1

Monitoring GUI (Sensor State - Visualization & Update)

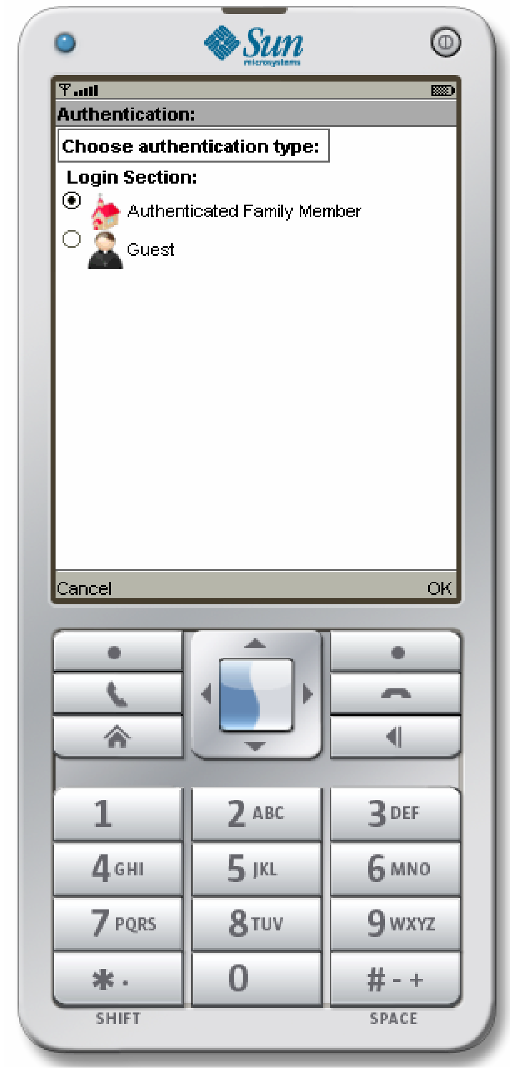


Sensor Type	NoiseSensor
Sensor ID	NoiseSensor2
Sensor Location	LivingRoom1
Sensor Value	MEDIUM
Sensor Status	ON
Timestamp	4465545656454

Smart House Demo



Mobile Agents Interface on Sun Java Wireless Toolkit

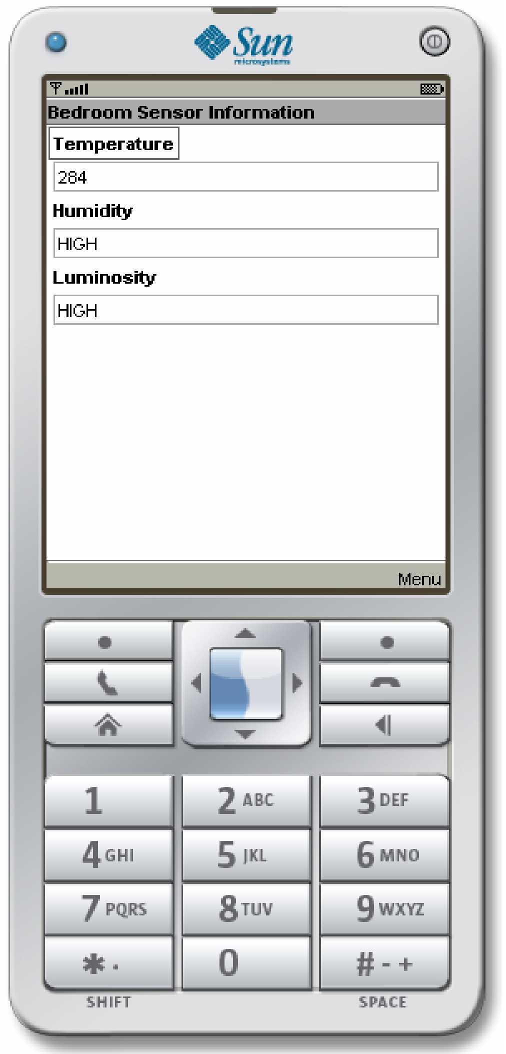

Authentication:

Choose authentication type:

Login Section:

[✓ Authenticated Family Member]

Guest


Bedroom Sensor Information

Temperature: 284

Humidity: HIGH

Luminosity: HIGH