Ontology Based Knowledge Formulation and an Interpretation Engine for Intelligent Devices in Pervasive Environments

Anna Magdalena Kosek

April 1, 2011

Edinburgh Napier UNIVERSITY

Submitted in partial fulfilment of the requirements of Edinburgh Napier University for the degree of Doctor of Philosophy in Computing

Contents

List of Figures

List of Tables

Abstract

Ongoing device miniaturization makes it possible to manufacture very small devices; therefore more of them can be embedded in one space. Pervasive computing concepts, envisioning computers distributed in a space and hidden from users' sight, presented by Weiser in 1991 are becoming more realistic and feasible to implement. A technology supporting pervasive computing and Ambient Intelligence also needs to follow miniaturization. The Ambient Intelligence domain was mainly focused on supercomputers with large computation power and it is now moving towards smaller devices, with limited computation power, and takes inspiration from distributed systems, ad-hoc networks and emergent computing. The ability to process knowledge, understand network protocols, adapt and learn is becoming a required capability from fairly small and energy-frugal devices. This research project consists of two main parts. The first part of the project has created a context aware generic knowledgebase interpretation engine that enables autonomous devices to pervasively manage smart spaces using Communicating Sequential Processes as the underlying design methodology. In the second part a knowledgebase containing all the information that is needed for a device to cooperate, make decisions and react was designed and constructed. The interpretation engine is designed to be suitable for devices from different vendors, as it enables semantic interoperability based on the use of ontologies. The knowledge, that the engine interprets, is drawn from an ontology and the model of the chosen ontology is fixed in the engine. This project has investigated, designed and built a prototype of the knowledge base interpretation engine. Functional testing was performed using a simulation implemented in JCSP. The implementation simulates many autonomous devices running in parallel, communicating using a broadcast-based protocol, self-organizing into sub-networks and reacting to users' requests. The main goal of the project was to design and investigate the knowledge interpretation engine, determine the number of functions that the engine performs, to enable hardware realisation, and investigate the knowledgebase represented with use of RDF triples and chosen ontology model. This project was undertaken in collaboration with NXP Semiconductor Research Eindhoven, The Netherlands.

Acknowledgements

This project have been initiated by NXP Semiconductors with cooperation from Edinburgh Napier University. I am very grateful to Aly Syed, Erik Lambers and Clara Otero for allowing me to join first the Digital Signal Processing and next the Distributed Systems Architectures groups at NXP Semiconductor. I appreciate the commercial experience, many interesting remarks and encouraging the project's movement.

I would like acknowledge Jon Kerridge, Aly Syed and Alistair Armitage for the guidance and help all though the project, excellent help with publications, interesting perspective, brain-storms and many afternoons spent drawing diagrams on a whiteboard or a piece of paper. I really appreciate this chance to work with excellent and very experienced researchers.

1 Introduction

The number of devices surrounding us increases every day. The devices that we use have become more complicated and have more capabilities and functions. A mobile phone is now used not only to make calls, but also to store and edit files, take photos and videos, play music, browse the web and much more. Some of these functions are repeated in many devices. Most of the devices we use are autonomous and making them co-operate with other devices is often difficult and time consuming. A mechanism to connect devices, make them co-operate and use each other's functions in a simple and transparent way would help to manage a network of every-day or commonly used devices. The method of making many computers physically available, but effectively invisible to the user is called pervasive computing. The concept was introduced by Weiser in 1991 and many researchers have been investigating it since. Pervasive computing is associated with research areas like distributed systems and mobile computing.

In pervasive computing the visibility of interaction between components is reduced to a minimum, with data and functionalities distributed. Pervasive computing uses transparent methods for organizing networks of devices, negotiating connections and collaboration. A pervasive environment is dynamic and components can be mobile. The ability to adapt to ever changing environment conditions and tasks assigned by the user is very desirable. Adaptation also involves self-organization, learning, reconfiguration and dealing with partial failures.

A pervasive adaptive system should be sensitive to the presence of people; it should adapt to their preferences and habits. This adaptation would have to, in an intelligent way, enable the system to react as users wish. The system would have to make its own decision and choices based upon observation and by sensing the environment. This kind of system will need many types of sensors and knowledge about the environment to make an sensible decision.

A pervasive environment should work with various sets of devices, and enable collaboration of the available devices. In the case of device failure, the system should reconfigure itself and continue to work with resources that are available. A central control workstation or remote control device is not the goal of the system. It is likely that a mixed control system will be required that is neither fully distributed nor fully centralized but the particular mix of control function needs to be determined and may vary with application environment.

The environment is a mix of fixed and mobile devices. Some devices can stay in the environment for a long period of time, some can visit a specific space only briefly. As a central control system is not present, devices in the network have to monitor the network topology and be ready to establish new connections or destroy existing links. Connecting devices is just the first step. The actual cooperation mechanisms and enabling devices or components to understand each other are challenging problems of pervasive computing. All components of the system have to understand a common language to enable cooperation.

A network of devices, from the same space, can be more useful for a user than a collection of autonomous devices. Enabling device collaboration by finding a common language and ways of communicating to perform a task assigned by an user is one of the main goals of this project.

1.1 Project Aim and Objectives

The project aim is to investigate the usability of Communicating Process Architectures (CPA) to design and develop pervasive and adaptive environments. A CPA uses CSP (Communicating Sequential Processes) [24] as its formal and theoretical underpinning. The project objectives are as follows:

- Explore the usability of CSP models to build a pervasive and adaptive system.
- Design a generic knowledgebase interpretation engine for devices from a pervasive system.
- Implement such an engine using a CSP implementation language.
- Design an ontology model for knowledge representation.
- Design and implement a knowledgebase using the ontology model that represents knowledge that devices need to cooperate in the proposed architecture.
- Improve the protocol for the architecture presented in [62].

- To undertake any experiments initially in a simulation environment (with a view to a hardware implementation) where such pervasive capabilities are required.

- Implement mechanisms to enable learning, emergent behaviour, forward and backward compatibility of devices using the presented interpretation engine.

The main goals are as follows:

- Integration of different platforms, components programmed using different languages and different hardware structures.

- Integration of small pieces of equipment into a large system to control lights in many building spaces.

- Demonstrate flexibility over a variety of devices by using a generic knowledgebase interpretation engine with a knowledgebase built on an ontology model.

- Use of sensors and actuators to adapt, in an intelligent and pervasive way, to the dynamic environment.

This project was undertaken in collaboration with NXP Semiconductor Eindhoven, Netherlands. From January 2009 NXP participated in the EU Artemis funded Sofia project (Smart Objects For Intelligent Applications) [1]. The project is investigating the use of pervasive systems in the application areas of Personal Spaces, Smart Indoor Spaces and Smart Cities. NXP's primary involvement is in the area of Architecture Development for Smart Indoor Spaces. This project is cooperating with Sofia, while sustaining its focus on CSP usability for pervasive systems.

1.2 Research questions

There are three research questions that this project is answering.

- **RQ1.** Is it possible to create a generic interpretation engine that reasons about knowledge presented in a machine readable form?

- **RQ2.** Is it possible to create a knowledgebase that contains capabilities, description and behaviour of a device in a simple, fixed, machine readable format?

- **RQ3.** Is a CSP based model suitable to represent a pervasive and adaptive environment?

The knowledgebase interpretation engine is intended to be developed in hardware, so the number of functions that the engine will perform has to be determined. Research questions referred to RQ1 are:

- **RQ1.1.** What is the set of functions that an interpretation engine requires?

- **RQ1.2.** For this type of systems can we identify a minimal set of functions that the engine requires?

1.3 Motivation

The project was proposed by NXP Semiconductors. NXP is a leading semiconductor company founded by Philips. NXP creates semiconductors, system solutions and software for TVs, set-top boxes, identification applications, mobile phones, cars and a wide range of other electronic devices [57]. One NXP research area is to improve control over equipment in the home environment by creating an intelligent network of devices. This project investigates the use of Communicating Process Architectures to create and control such an environment.

1.4 Academic Collaboration

This project was supported by PerAda, Future and Emerging Technologies Proactive Initiative on Pervasive Adaptation (*www.perada.eu*), which is a Seventh Framework Programme (FP7) funded by European Commission CORDIS, Community Research and Development Information Service (*cordis.europa.eu*). PerAda is a research network which aims to bring researchers together to discuss and share ideas relevant to Pervasive Adaptation. PerAda provided funding for collaborative research between Edinburgh Napier University and NXP Semiconductor that was crucial in the co-operation involved in this project.

1.5 Commercial Collaboration

This project is associated with the Sofia project [1] funded by ARTEMIS-IA, Advanced Research and Technology in Embedded Intelligence and Systems Industry Association (*www.artemisia-association.org*). The findings from this project contributed to deliverables for the Sofia project on domain-specific ontologies. Work on lighting and sensor ontologies was done in cooperation with Technische Universiteit Eindhoven (*www.tue.nl*). The work on representing rules in RDF format is being considered to be used in Sofia's semantic information brokers (SIB) for reasoning purposes [1]. The Knowledgebase Interpretation Engine was submitted as a hardware patent and it is being reviewed within the NXP Semiconductors Intellectual Property Department.

1.6 Structure of thesis

The structure of this thesis consists of six main parts. Chapters 2, 3 and 4 present the background to the project and outline the research area presented in this work. Chapter 2 describes different domains that are relevant to the project, Chapter 3 presents the chosen concurrency model used to simulate an environment. The outline of research is continued in Chapter 4 that presents various knowledge representation techniques. The second part, consisting of Chapters 5 and 6 presents the chosen architecture and outlines a scenario selected for the architecture.

The main part of this thesis are Chapters 7 and 8 that represent the main research focus of this project. Chapter 7 introduces and describes the knowledge representation chosen for the project and Chapter 8 describes the reasoning mechanism interpreting this knowledge. Chapter 9 presents experiments with the architecture, knowledge representation and reasoning component following the scenario and case studies as presented in Chapter 6. Finally in Chapter 10 conclusions and ideas for further work are presented.

2 Theoretical Background

Chapter 2 describes the theoretical foundations of the research of the project. The pervasive computing research area is based on distributed systems, mobile computing and adaptive systems. Mechanisms associated with adaptation, reconfiguration and learning are closely associated with artificial intelligence and ambient intelligence. This chapter provides a review of these domains to draw a bigger picture of the research area of the project.

A smart home, office or warehouse space needs to be supported by a computing infrastructure that needs to deal with many devices with different functionalities and enable communication and co-operation. In existing systems processing power and a storage space of a computer was sufficient for a control system and repository to support a few devices. Nowadays the number of devices in a space grows rapidly and device miniaturisation enables manufacturing very small devices that can be deployed in a smart space. As people using smart spaces would like to control many devices and get specific and accurate data from as many sensors as possible, the central controller and data repository is under a stress of processing many connections and large amounts of data. The central unit creates a *bottleneck* in the system and requires a powerful computer to process all the data gathered by a smart space. A fully centralised architecture is not a good solution for large scale control systems. More distributed and clustered approaches can be used to take the load off the central unit.

2.1 Distributed Systems

Distributed computing refers to programs that make calls to other, than its own, space addresses on other machines [63]. Therefore hardware and software components from computers on a network co-ordinate their actions by communication [13]. A distributed system consists of many machines with different architectures and locations connected by a network. The resources are distributed, so the system may call other machines. The problems associated with distributed computing arise from the fact that nothing, except support of a particular interface, is known about the receiver of the call. The client does not know the hardware architecture or the language the recipient's machine was implemented with [63]. Distributed systems have to take into account latency, differences in models of memory access, issues of concurrency and partial failure [63].

2.2 Interoperability

A well known problem in distributed systems is how to achieve interoperability between components in the system. There are three classic interoperability levels: platform, programming language and service interoperability [61]. Service interoperability divides further in to signature, protocol and semantic levels [61]. The signature interoperability problems are associated with the syntax of the interface of the service and are successfully solved by popular languages interface specification like CORBA's IDL (Interface Definition Language) [47] or WSDL (Web Service Definition Language) [11]. Many network interoperability problems (in protocol level of interoperability) can be addressed by Internet protocols, however these do not solve interoperability problems on a semantic level [19]. When considering many multi-vendor devices and components, solutions for semantic interoperability based only on standards do not scale [48], flexible and updatable standards are needed.

2.3 Mobile Computing

The appearance of mobile computers and wireless local area networks created problems with building distributed systems with mobile devices. Mobile computing needs to consider the performance of computing tasks while a user is moving or in an environment that is other than its own [13]. Although the user is away from its own local network it is still able to access resources using devices they took with them. In the new location a mobile device can still use the Internet or access nearby devices which are defined as context-aware and location-aware computing.

The idea with mobile computing is that although a user is moving and changing their physical location with a mobile device, the network supporting co-operation appears to be stable. The device needs to switch between connections, use available resources and guarantee that it can always provide the service that the user is requesting. This approach requires self-configuration, service discovery, decision making capabilities and context and location awareness from the device. Other problems that this area of research meets are differences such as network quality, reliablilty of mobile elements, limitations of local resources caused by size and capabilities of devices and battery power consumption [55].

2.4 Pervasive Computing

The pervasive or ubiquitous computing concept was first described by Weiser in 1991. The main concept of this approach is to make many computers physically available, but effectively invisible to the user [67]. Devices, that the system consists of, are in various sizes, have different capabilities and functions, each suited to a different task. The term *pervasive* suggests that devices will finally become so pervasive in everyday objects, that they are hardly noticed [13].

Devices in pervasive computing systems are enabled with communication capabilities and are designed to be useful in a single environment such as home or a hospital. The idea is to integrate all the devices that are connected to the network to cooperate and use each other to achieve an expected performance and capability. In this system computers are no longer tools but assist humans in their everyday activities. All the computers in the environment will not be considered as autonomous but parts of a bigger system that is targeted to users' needs. Devices in pervasive systems have to be aware of their location and surroundings. The computer knowing of their position can adapt its behaviour to the environment [67].

Pervasive computing is based on distributed and mobile computing and defines additional problems to be solved: effective use of smart spaces, invisibility and localized scalability [55]. The decision making process is based on users preferences and should be controlled to meet user' expectations.

2.4.1 Effective use of Smart Spaces

A smart space is a well defined area, for example a room, corridor or courtyard with all the devices available in it. The smart space is controlled by a computer structure embedded in the building infrastructure [55] centred or distributed. The smart space has to be equipped with various sensors, so it can recognise changes appearing in the defined environment. A simple example of smart space is an automatic adjustment of cooling, heating and lights levels in the room depending on occupant's preferences. Not only residents of the smart space can manipulate it, but influence from other directions is also possible. A smart space's context can be applied to devices that are visiting the space. Software on users' devices can behave differently depending on location. For example all mobile phones can be switched to silent mode when entering a theatre. The device can also have more functions whether located in a smart space or not. A capability of controlling various equipment in the space can appear or disappear depending on a user's location.

2.4.2 Invisibility

An ideal pervasive system described by Weiser disappears from user's sight [67]. In practice this will be achieved with minimal user distraction. If the system continuously meets users expectations and rarely makes wrong decisions which surprise the user, it can disappear from a user's consciousness [55]. Therefore a pervasive system has to make decisions without asking users. The knowledge should be based on a user's behaviour and preferences captured previously. The pervasive system should maintain user profiles and use them to make its performance more invisible.

The other factor of making a computing system pervasive is the actual size of devices. With device miniaturisation concepts like Smartdust and Unity Fog becoming more realistic. Smartdust [31] is a concept of a network of autonomous nodes that have capabilities of sensing, computing, communicating and posses a power source in a square millimetre volume. The devices described in this concept are so small, they can be spread in a space like a dust. Unity Fog [21] is a swarm of nano-robots that can make shape of anything and quickly change into another shape.

Intelligent miniature devices are desired to be components of pervasive systems. In order to create Smartdust and Unity Fog, co-operating networks of devices have to be equipped with very small chips/engines to enable computing, reasoning and communicating. These networks of very small devices need capabilities of reconfiguration and self-organisation because manual configuration is not possible due to the number and the size of devices. The *install and forget* approach is the most desirable when it comes to deployment and maintenance of a dust-like network of devices.

2.4.3 Decision Making

It is crucial for the pervasive system to track user's intent. Actions preformed by a system should help rather than hinder the user [55]. Generic applications have no idea what the user is attempting to do so can offer little support for adaptation. On the other hand there are systems that annoy users with too many questions and hints. To solve this problem, the user's intent should be inferred from other sources. This can be achieved by taking into consideration statically specified user profiles and information obtained through dynamic interactions [55]. To decide what will influence the decision the most, is a very difficult task. When considering office spaces there are two factors that are important: the users' satisfaction and policies deployed by the building owner or a company. As the system should help a user to feel comfortable, it should also respect rules set by the maintenance team. This could include energy saving solutions, carbon footprint minimisation and other policies that might effect office automations. The decision making components need to take under consideration many facts and prioritise while making pervasive decisions that satisfy many rules.

2.4.4 Localized Scalability

The number of devices in a smart space and users using the defined location should not be limited. Therefore the number of interactions between a user and its surrounding increases. As a result scalability is a problem for pervasive systems [55]. In ordinary networks a web server or a file server should handle as many operations as it is possible, regardless of physical distance. While considering smart spaces the priority should be given to the users that are in the environment, rather than remote customers.

The scalability problems often appears when there is one device managing the decision making process and control devices in the network. Some systems are using a distributed approach to extend scalability and get rid of the *bottleneck* of a single controller, and grouping is applied to control a set of devices.

2.4.5 The Gathor Tech Smart House

An example of pervasive system developed and used in home environment was presented in [22]. The Gathor Tech Smart House is an environment specially designed to improve everyday life of elderly people. The smart house provides an assistive environment that can sense other devices and residents. The house is equipped with many gadgets, for example: smart front doors recognising residents by Radio Frequency Identification (RFID) tags; smart mirror displaying important information or reminders; smart blinds controlled by a remote device; smart floor that senses residents location, so information about their habits can be stored [22].

The system designed to support the Smart House has a layered architecture, comprising physical, sensor platform, service, knowledge, context management and application layers. Those layers are part of a generic reference architecture applicable to any smart space [22].

One of the most interesting parts of the project is a smart plug. Most of the devices on the market do not contain a controllable interface or use protocols that are incompatible. Smart plugs provide an intelligent way to sense electrical devices in the intelligent space [22]. Every socket in the house is equipped with RFID (Radio-Frequency IDentification) reader and the devices' plugs have an RFID tags describing the device. This way, the system, at low cost, can recognise and install and subsequently identify to the smart space every new piece of equipment. Therefore the system has an ability to evolve as the new technologies emerge or as an application domain changes [22]. The system supporting the Gathor Tech Smart House uses simple solutions like low cost RFID tags to provide machine readable device profile to enable interoperability between multi-vendor devices. As devices arrive in the space, the tag can be red and if the computer infrastructure supporting the smart space recognises the device it can be used and integrated into the Gathor Tech Smart House. A problem with this approach appears when a device is so advanced and new that it is not recognised (without an update in the controller), therefore it cannot be used and cooperation with other devices is not possible.

2.5 Adaptive Systems

The ability to adapt is an important requirement for a pervasive environment. New services, functionalities, interaction mechanisms or devices can be added to the pervasive system requiring them to be adapted to the specific characteristics of the environment [51]. Mobility of devices makes system's network topology even more dynamic. Mobile devices have to be able to detect change of location and exploit knowledge about their current situation, called location-awareness [33]. All mobile devices have to detect and react to the environment and therefore adapt to a new space to improve quality of communication [33]. On the other hand existing elements from the system might be adapted to user's requirements or changing conditions in the environment [51]. This characteristic of pervasive systems is called context-awareness.

An ability to react to a context change is important in interactive applications. Context can be defined as implicit situational information [14]. In human-computer interaction it is important that the computer understands context for better communication between a machine and its user. In mobile computing a user can travel with their device to different places, therefore context information such as location, presence of other people and objects, environmental conditions, can be dynamic. Devices need to adapt to changing context and offer services appropriate and possible in a particular environment and situation. The primary context types are: location, identity, time, activity [14], computing environment (e.g. available devices and services) and physical environment (e.g. light intensity, noise level, humidity) [56]. Context-aware applications need to be sensitive to change of context over time [14]. An interactive system needs to react to signals, a pervasive system needs to make a conclusion from many signals to deduce context and react to it in a seamless and invisible way.

Applying adaptation to a pervasive system is a challenging task, mostly because it has to adapt to change in multiple elements like devices and services [51]. The system is required to function without central control so the adaptation has to be applied to many devices and cannot be controlled just by one device.

Wireless systems have to be able to adapt to the different opportunities and limits of different types of mobility. Low mobility will be appropriate mostly for pedestrians or easily reconfigurable desktop equipment, where high mobility can be suitable for moving vehicles [33]. It is important to distinguish these two types of mobility. The pervasive system with wireless links is likely to be a hybrid with high bandwidth in areas like buildings or airport and lower bandwidth in rooms or well defined smart spaces.

2.6 Sensors

A sensor is a piece of hardware that measures analogue physical parameters such as pressure, temperature and light level [65]. Sensors have always been an essential part of computing. The keyboard is a simple example of set of pressure sensors activated by keys. This way a user can communicate and send commands to an operating system. Other examples are: mouse, microphone, camera or touch screen. Now more complicated sets of sensors are being added to our computers. The increasing number of sensors that computers use requires more processing power for data manipulation and more memory space to store information from sensors. The National Institute of Standard and Technology Smart Space Project designed and built a context-aware smart meeting room that sense ongoing human activities and respond to them [60]. The architecture consists of 280 microphones and 7 HD video cameras, that produces 200 Gb per hour of data that support people recognition [60]. Therefore a computer that can store a huge amount of data from sensors and process them has to have sufficient processing power and memory space.

In the project presented in the following chapters the network consists of many devices and many sensors of different kinds. One computer cannot be responsible for processing all the signals from the system. A mixed control system is the aim of this project. Therefore sensors can be grouped by location, sensor type or purpose. Grouping sensors can help customizing space and making decisions for a particular environment.

Many environments, even if they are treated as separate spaces, can be influenced by other environments. The controlling system responsible for a particular location has to be aware of signals in its own space as well as other environments influencing it. The whole picture of an environment can be the key to making accurate decisions.

Mobility of devices can often change their context of use. Automatic customization is called context-aware operation [65]. For mobile devices it is important to recognise the environment. Devices can use their own sensors to determine tilt angle, whether is shaken or moved [65]. The location aware device can also use sensors, or information obtained from them by a location control system, to customize itself and adapt to the environment or users' needs.

Sensors are very important part of an adaptive system. Most of the changes in an observed environment should be noticed and adequate action should be performed. A system that adapts to user needs and expectations has to rely on signals received from sensors.

2.7 Ad-hoc Networks

The devices in pervasive system have to be enabled with communication. The set of devices creates a network that may have various topologies and parts of it can be mobile. Managing dynamic networks is a difficult task and many researchers have been solving problems associated with it [12, 15, 39].

In wireless communication systems there is a need for creating nets of independent mobile nodes. This kind of network will have various numbers of nodes and a dynamic topology. The communication between nodes cannot rely on some central control mechanism so any number of nodes needs to be able to create a network and co-operate. These kinds of decentralised networks have to consist of nodes that are able to recognise a network topology, other nodes' availability and capabilities. These kinds of networks are called ad-hoc networks and defined as a collection of mobile and fixed devices communicating over wireless links [15]. A mobile ad-hoc network should be a set of devices that recognise each other, decide about inner connections, organise a virtual topology, exchange and use resources and capabilities. The network should appear to be stable, and applications should be unaware that it is changing under them [15].

Designing network protocols for mobile ad-hoc networks is a very difficult, complex and challenging issue. These kinds of networks require efficient algorithms to determine network topology organization, connections and routing. The network should be able to reconfigure itself and to determine if it's able to continue to function. Some nodes can be disconnected or fail and also new nodes can be connected to the network. The network mechanisms have to customise a topology to available devices.

The constructed network can consist of many similar devices, with comparable or identical capabilities. Devices can be also localized in one area: the same room or floor of the building. Those kinds of devices can be grouped. Elliott and Heile [15] present a model of an ad-hoc two level hierarchical network (Figure 1). Nodes are divided into groups and organized into clusters. Every cluster has a cluster head which is the chosen node of the group of nodes. The network diagram is shown in Figure 1. The head node is one of the nodes from the group, and it is not physically different.

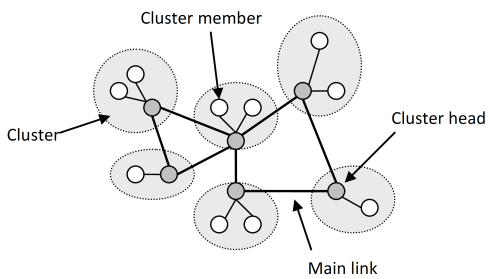


Figure 1: A hierarchical ad-hoc network diagram [15].

This chosen node is created to represent the cluster and allocate tasks to the nodes that are in the cluster if necessary. Every node has to be able to become a head of the cluster, and this function can be switched between nodes. Cluster heads from different groups are connected, and the knowledge about particular group members is local, therefore only the head of the cluster knows its cluster's topology.

A two level hierarchical network model helps to balance network capacity and delays [15]. While searching for a particular node only clusters heads can be investigated, and if it matches, a local cluster can be searched.

2.8 Ambient Intelligence

The network of devices in pervasive systems is required not to have a central control system, therefore the devices have to make decisions, co-operate and adapt to new situations. The pervasive adaptive environment for home, office or warehouse environment is required to be aware of the people presence and adapt to their preferences or needs.

Ambient Intelligence (AmI) refers to electronic environments that are sensitive and responsive to the presence of people. The aim of ambient intelligence is to create an environment that assists people in their everyday activities. AmI adapts to people not the other way around, its design starts from studying person to word interactions. The aim of AmI is to create secure, reliable and calm communication between users and computers [4]. Where pervasive computing and ad-hoc networks are about integrating devices, AmI adds to an environment people that the system has to adapt to.

The main research domain of the project is pervasive computing, but there are many domains associated with this subject. To create a support for a smart space the software or hardware infrastructure needs to support mobility (Section 2.3), resource and function distribution (Section 2.1), deal with self-configuration and reconfiguration in a pervasive way (Section 2.4). Decision making process (Section 2.4.3), interoperability (Section 2.2) and scalability (Section 2.4.4) issues need to be addressed and solved in a smart space. The pervasive system need to be aware of context, location and the environment (Section 2.5), adapt to user needs while obeying some high priority rules.

3 Communicating Process Architectures

The presented network of devices with distributed functionalities and dynamic architecture can be viewed from three different perspectives, as presented in Figure 2. The view **A** (Figure 2) is called a top level and in this view the system consists of many devices, running in parallel and performing some behaviours. The specialisation of the view **A** is view **B**, where only a set of devices grouped in a sub-network is visible, this view is referred to a virtual-device and explained in detail in Section 5.3. The third view **C** (Figure 2) is a view restricted to a single device and called the bottom level view.

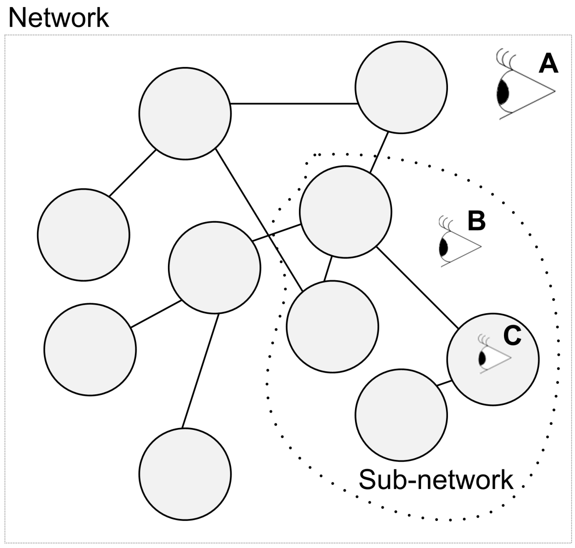


Figure 2: Different views of the network of devices.

In the top level the system is an asynchronous, complex and distributed environment with non-deterministic behaviour. The operation of the system has to be expressed with a parallel software infrastructure that does not rely on a global synchronisation, therefore an asynchronous parallel programming language must be used [64]. Languages used for high-level modelling of synchronous systems are often based on Communicating Sequential Processes (CSP [25]) [64]. The system supporting a pervasive environment does not need any global synchronisation and each element proceeds at its own speed, the only synchronisation that is needed is on messages, therefore synchronous CSP channels are used. The project aim is to investigate the usability of Communicating Process Architectures (CPA) in pervasive adaptive environments. CPA describes an approach to process-oriented system development. It is based on the theory of Communicating Sequential Processes [25].

This chapter describes fundamentals of CSP and parallel systems. CSP is based on a mathematical model, therefore it is possible to prove correctness of a designed system. In section 3.2 available verification tools are described. Different CSP implementations are listed and described in section 3.3. Finally the topic of targeting a hardware design using CSP is presented in Section 3.4.

3.1 Communicating Sequential Processes basics

Communicating Sequential Processes (CSP) is a formal language consisting of mathematical models and methods to construct component processes to interact with each other by communication [54]. Communicating Sequential Processes were first described by C.A.R Hoare in 1978 [24] and then explained in more details in his book [25] representing a mathematical notation for concurrency theory.

3.1.1 CSP Fundamentals

A parallel style of programming enables the creation of a system with processes working concurrently. In parallel architectures processes run simultaneously and are connected with channels to enable communication. One of the main advantages of CSP system design is that simple processes can be composed into larger networks [25]. In computer science this approach is called separation of concepts. Where concepts can represent features or functionalities of the system. In this design approach functionalities are implemented separately and composed into a larger architecture.

CSP applications are process-oriented. A system consists of processes that are sequences of instructions. Process run separately and can communicate with other processes using channels [25]. Therefore it is easy to compose processes into networks using channels to enable communication. Processes from one network can work on different processors or even communicate across different devices. A channel is a point-to-point connection between two processes [25]. A simple version of a CSP channel consists of an in-end and an out-end performing a one-way communication. One process writes to a channel and the other reads from it. To establish a connection it is necessary to have at least two processes, one process is connected to an in-end of the channel and the other process is connected to out-end of the channel as it is shown on Figure 3.

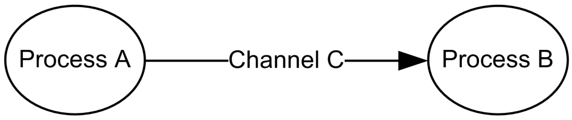


Figure 3: A simple example of a CSP network diagram.

On a diagram (Figure 3) a channel is represented by an arrow to show the direction of the communication. If bidirectional communication is needed an additional channel in the opposite direction should be added. Channels are unbuffered, so process A (Figure 3) writes to the channel C and waits in an idle state for process B to be ready to read. In this way channels are unbuffered and communication occurs only when both sides are ready. This simple channel is constructed in a way that data sent over it cannot be lost. In CSP based systems buffer space is not required. That makes a system design simpler, robust and easier to calculate memory needs.

3.1.2 Timers

Many systems rely on time. Timers are fundamental components of most of the programming environments. Time is delivered from the processor's clock. Processes can be made to stay in the idle state for some defined period. Alarms on timers can be set and detected so some actions can be scheduled. For example a process can be designed to send a signal to other processes every minute to update its status.

3.1.3 Alternative

When processes have many channels and the order of an input communication is random it is possible to distinguish from which channel a communication appeared. The alternative (ALT) programming structure captures channels behaviour and permits selection between one or more input communications. The alternative construct is assigned with the word ALT and consists of guards that can be channel ends or timers [2]. The structure of alternative is presented below:

ALT (Guard 1,
(Guard 2,
...
Guard N)

As soon as one guard becomes ready then it is chosen from the list of guards and instructions associated with it are carried out. Alternative can choose only one guard from the list, if many guards are ready it chooses one of the ready guards according to some selection criteria. If none of the guards from ALT is ready than the alternative waits in an idle state till one of them becomes ready.

3.1.4 CSP design concepts

Most of the processes in parallel systems are designed to run without termination. This way processes can wait for a communication from different processes or scheduled tasks. Some of the process can terminate after all their tasks are finished and further communication with them is not possible.

Concurrent systems might be more challenging to design and understand than sequential. In sequential systems at one time just one line of code is executed, and events happen one after another. In parallel systems there can be many processes working at once and they can influence one another. It is necessary to understand process states and design a system to avoid problems connected to the consequences of process states and influences. Processes can influence each other and communicate. There can be much more specific misbehaviors and designs have to take them under consideration [54].

Another problem in concurrent systems is nondeterminism. A system is nondeterministic if two different copies of it can behave differently when exactly the same input is given [54]. Concurrent systems can behave this way because of many aspects. One of the reasons can be the order of the communication between processes. If a system's performance depends on the order of the communication, its behaviour can be different every time it is executed. In this manner a system's performance cannot be controlled or tested precisely, however individual process behaviour can be tested.

A system is deadlocked if any process cannot make progress because it is waiting for a communication with other processes. The most popular example of a deadlock is Five Dining Philosophers. Five philosophers are seated at a round table with a single fork on a right-hand side [25].

Every philosopher needs two forks to eat, so they have to borrow a fork from the philosopher from the let-hand site. If they start to eat simultaneously and pick up their left fork then they deadlock and starve to death. None of the philosophers will have a chance to pick up the right fork. Although the example is very simple it shows one of the most important causes of deadlock: competition for resources [54]. Deadlock is one of the biggest problems in parallel systems.

Most programmers are familiar with programs that go into infinite loops, in which state interaction with the environment is impossible. If a process performs an unbroken infinite sequence of internal actions the system is livelocked. This happens when a network of processes communicate internally without an external communication [54]. Livelocked system can look similar to deadlocked, but both of those states are very different.

Client-Server design pattern The deadlock and livelock can be avoided by design a CSP based system. If the design fulfils the Client-Server design pattern it is deadlock free. In the Client-Server design pattern the client sends a request to the server. The Client having sent a request to a Server guarantees to read any response immediately. The Server has to accept any request immediately and respond in finite time. Additionally a server will never send a message to any of its clients without a request from the client. Depending on a request the server may or may not send a response to the client. The server can behave as a client to other severs. Considering one communication link, there should be a server and a client at each end of the link. Labelling a client and server ends of communication is a way to determine architecture correctness by ensuring there are no cycles of clients and servers within the network.

3.2 System verification

Concurrent systems can cause many problems, so it is necessary to have a good understanding how they behave and how to accommodate nondeterminism and avoid, livelock and deadlock during the design stage of the system [41]. A tool for software verification would be useful for testing purpose and improve software reliability.

Failures-Divergences Refinement (FDR) is a model-checker for CSP designed to asses correctness conditions including deadlock and livelock freedom and also general safety and liveness properties. FDR is a product of Formal Systems (*www.fsel.com*).

Spin is another verification tool and is an open-source software tool that can be used for formal verification of concurrent and distributed software systems [26]. Spin can be used both as a simulator and a verification tool. A Spin model is a logic model checker, and accepts a specification language called Promela (Process Meta-Language). Importantly, Spin incorporates a concept of mobility in its design that makes its use more appropriate for the mobile distributed systems that can occur in pervasive adaptive systems. The mobility is described with use of a process calculus, called π-calculus [44]. Moreover a Spin model can verify a system as a whole, including dynamic and static sections. This capability is useful when checking if a system with a dynamic topology is deadlock and livelock free.

A pervasive adaptive environment can be built using components of a system working concurrently. It is therefore reasonable to question whether a CSP model can be used when building a pervasive adaptive system. As the deadlock and livelock can be avoided by design and software can be verified using a Spin model or FDR checker, the advantages of the CSP model can be used when designing and implementing a pervasive adaptive system. Every device from the system can run processes communicating with processes on different devices. Processes on one device can be grouped in sets and cooperate. Every process can be responsible for different device capabilities, communication or synchronisation with different devices.

3.3 CSP implementations

There are many languages that implement CSP model. This section describes JCSP, CTC++, occam, occam$-\pi$ and POOSL and justifies the choice of a language used to simulate presented architecture.

3.3.1 occam and occam$-\pi$

`occam` is a programming language implementing the CSP algebra that was originally used for programming Transputer microprocessors [49]. `occam` is the programming language that is the most widely used when representing CSP models. The `occam`$-\pi$ is an extension of `occam` implemented for KRoC, the Kent Retargetable occam Compiler CSP kernel [69]. The `occam`$-\pi$ implements the π-calculus, process calculus, enabling expressing models of systems that change their architecture during runtime [44].

3.3.2 Handel-C

The Handel language is a hardware description language that is a subset of `occam` and used for hardware synthesis. A compiler was created to translate Handel to hardware Xilinx FPGA's [58]. The Handel language has evolved into Handel-C, a subset of C extended with features of parallel composition and synchronous channels [58]. Automatic translation of Handel-C to FPGA's is done with use of DK Design Suite developed by Mentor Graphics (*www.mentor.com*).

3.3.3 CTC++

CTC++ is a C++ library implementing CSP principles, created in University of Twente [8]. A tool developed at the University of Twente gCSP uses a graphical interface of CSP diagrams to generate CSPm (CSP algebra representation for FRD checker), CTC++, Handel-C and occam code. The gCSP proved to be a very valuable educational tool, but while still being under improvement is not suitable for the development.

3.3.4 JCSP

JCSP is an environment allowing the development of Java applications following CSP principles. Use of a Java based language enables executing designed processes on any device that can run the Java Virtual Machine (JVM), because of the cross-platform nature of this language. The use of Java also gives the advantage of a very popular and mature language with many libraries developed to help in a programming process. The advantage of using a mature language is that Java programmers can easily start programming with JCSP with the use of *jcsp* (current version 1.1rc4) libraries [68].

3.4 Targeting hardware with CSP

There are many concurrency models, this research investigates usability of CSP for building and simulating a pervasive environment. The CSP model was an inspiration to build a Transputer in the 80's. A Transputer was a high performance microprocessor developed by Inmos Ltd. now taken over by STMicroelectronics (*www.st.com*) that supports parallel processing through on-chip hardware [27]. The Transputer was intended to be programmed with `occam`. CSP model have been used to develop and design hardware with use of CSP‖B [43], Balsa [3], CTC++ translated into Handel-C with use of gCSP tool [7, 8, 29], JCSP [34]. An automatic tool to translate CSP diagrams to generate gate-level synthesis have been implemented [64, 58]. This project uses CSP concepts when designing all levels of the presented system, as shown in Figure 2, and represent a reasoning component expected to be built in hardware, as presented in Section 8, with the use of JCSP for simulation purposes.

4 Knowledge representation and ontology

With fast technology development it will soon be possible to deploy sensors and actuators everywhere. Many every-day use devices will have sensing, processing and wireless networking capabilities. To enable these devices to make decisions, knowledge and reasoning capabilities are needed. This chapter and Chapter 7 presents techniques used for knowledge representation and formulation, Chapter 8 presents the reasoning component designed and developed in this project.

In Artificial Intelligence (AI) a key to building powerful and large systems is capturing knowledge in a computer-readable form. Representing knowledge is a complex and time-consuming task. Therefore knowledgebase construction remains one of the biggest costs when constructing an AI system [46]. Most complicated AI systems require building a knowledgebase from scratch. Ontologies can offer a core structure for representing knowledge and enables reuse of knowledgebases when designing systems in similar domains. This chapter describes the research area of knowledge representation and knowledge engineering in Section 4.2. First the concept of an ontology and its specification for particular domains is presented in Section 4.1.

4.1 Ontology

An ontology is a representation of a set of concepts, functions and relations between concepts [20]. Concepts are classes of entities existing in a domain, functions and relations are respectively one-to-one and one-to-many properties existing in the environment. An ontology, together with syntax and semantics provides a language that is necessary for proper communication. Building an ontology is a difficult task because domains and language always evolve. It is impossible to state that the ontology describes all possible concepts and relations in a specific domain, but it can present a very up-to-date picture of the domain.

4.1.1 Ontology use for interoperability

Because an ontology can always evolve and adapt to an ever-changing domain, it can be a very flexible and up-to-date description of the domain. Therefore an ontology can play the role of a flexible standard and can be used to achieve semantic interoperability.

It has been shown that interoperability can be achieved by using an ontology in consumer electronics devices [62]. Another project, Sofia [1], is targeted to enable and maintain cross-industry interoperability for smart spaces using a core ontology based on DOLCE [42] and domain ontologies. The approach presented in this project is an extension of both [62] and Sofia concepts to create a pervasive environment with interoperability at the semantic level and re-usable knowledge representation based on an ontology. The specific environment that this project is considering is a network of very small, energy-frugal devices with limited processing capability. Services that are offered by these devices are simple, therefore it is possible to represent them using an uncomplicated knowledgebase drawn from an ontology.

4.1.2 Foundation ontology

A part of the knowledge in the system comes from understanding general concepts. Therefore a description of general truths is needed. The foundation or upper ontology describes general concepts and relations that are applicable to all knowledge domains. There are many upper ontologies, as there are many views of the world and concepts existing in it. This project is following the Sofia approach by choosing DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) as its upper ontology.

DOLCE ontology The DOLCE, Descriptive Ontology for Linguistic and Cognitive Engineering, is an upper ontology which is the first module of the WonderWeb Foundational Ontologies Library (WFOL). DOLCE was defined as a part of the OIL, Ontology Infrastructure for the Semantic Web [16], ontologies are applied to the World Wide Web to create the Semantic Web. The Semantic Web is a concept of a machine readable representation of the information on the World Wide Web

[5].

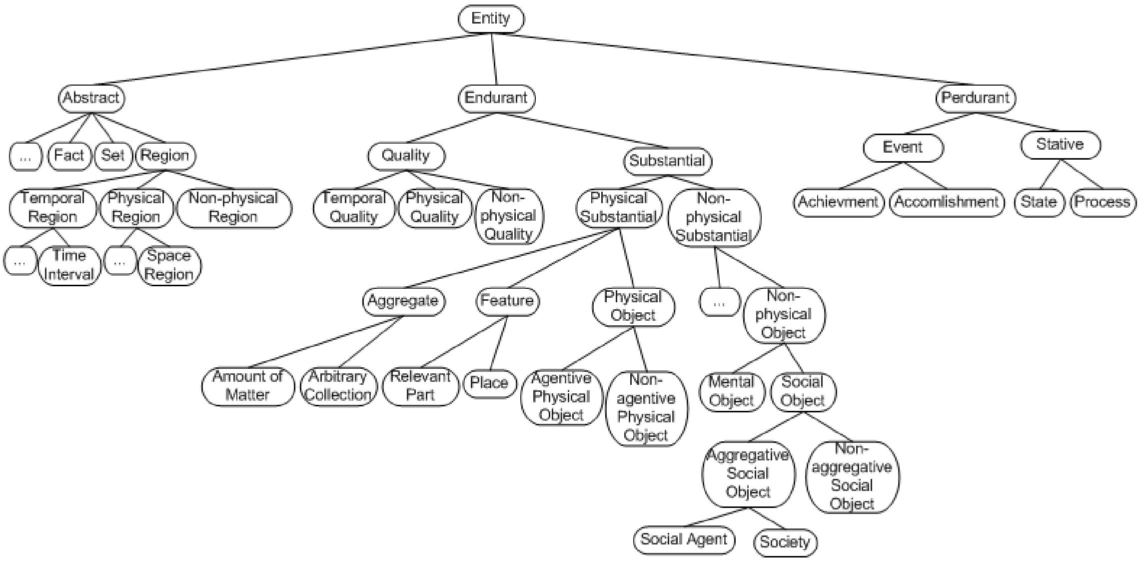


Figure 4: Taxonomy of DOLCE basic categories [18]

The DOLCE ontology divides the world into endurants, pradurants and abstract entities as presented in Figure 4. The difference between perdurants and endurants is their behaviour in time. Endurants are fully present at any time of their appearance. On the other hand perdurants are only present partially, therefore some of their temporal parts, previous or future phase, may not be present. Therefore endurants are entities that are present and perdurants are entities that happen in time. For example a person, that is an endurant, can participate in a discussion, that is a perdurant. The third main category of entities described by a DOLCE ontology are abstract entities. Abstract entities have no spatial or temporal qualities and they are not qualities themselves. An example of an abstract entity is a physical space, like a room or a park.

The DOLCE ontology is used to structure a general knowledge, so domain and application-specific ontologies can be attached to it, creating a graph of a knowledge.

4.1.3 Domain-specyfic ontology

Apart from understanding general concepts, a device needs also to have knowledge about concepts and relationships in its own domain. Domain-specific ontologies are designed by experts in the field and can change over time with advances in the domain. An example of domain-specyfic ontology is a lighting ontology developed as a part of this project.

Lighting ontology The lighting ontology was inspired by the description of lighting domain in IFC (Industry Foundation Classes)[9]. The IFC is a data format for describing, exchanging and sharing information in the building industry sector. The *IfcLightSource* IFC class is presented in Figure 5.

The lighting ontology was developed with the use of concepts proposed by IFC to represent knowledge about a light source. The lighting ontology is presented in Figure 6.

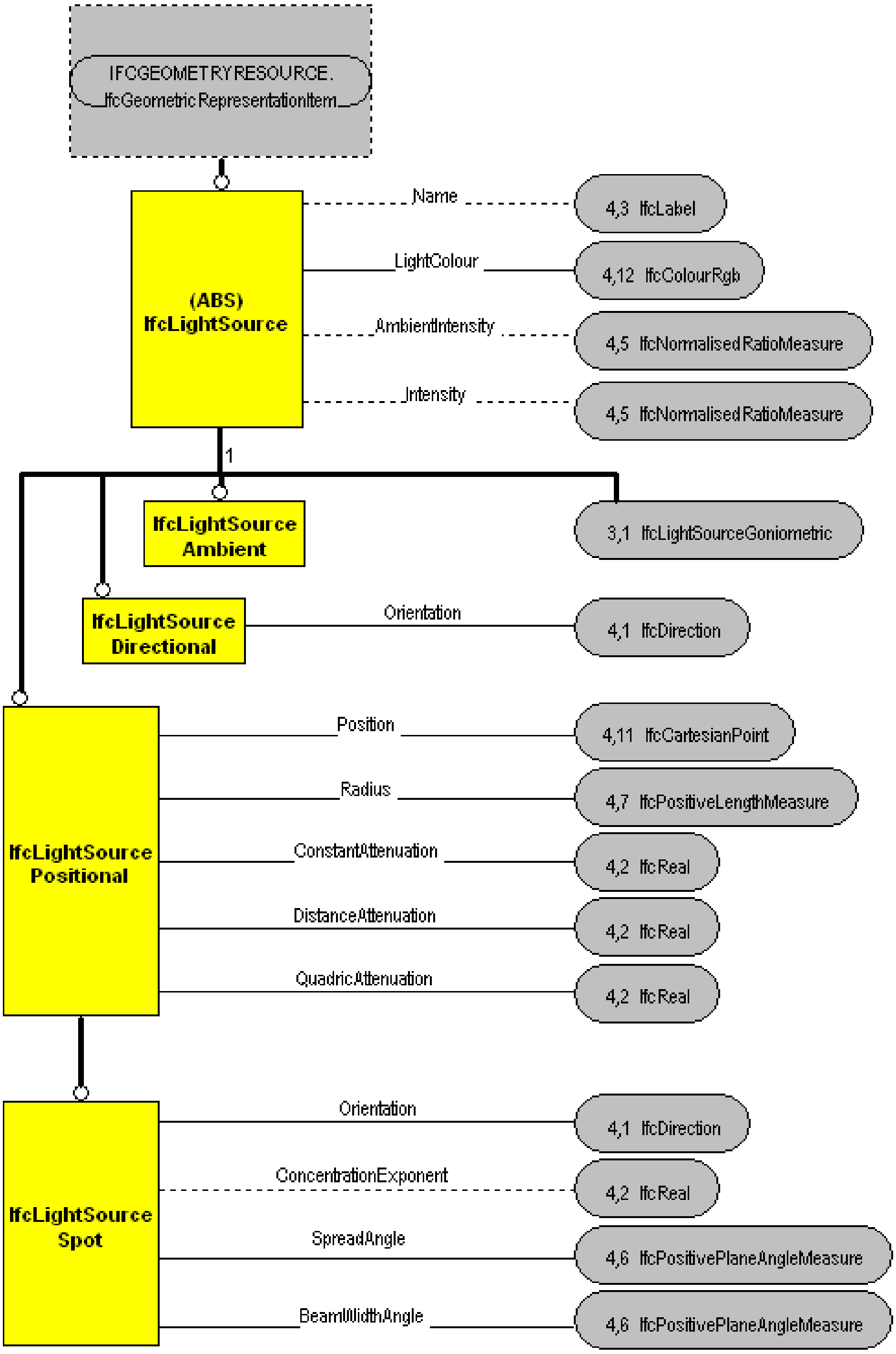


Figure 5: IFC light source description.

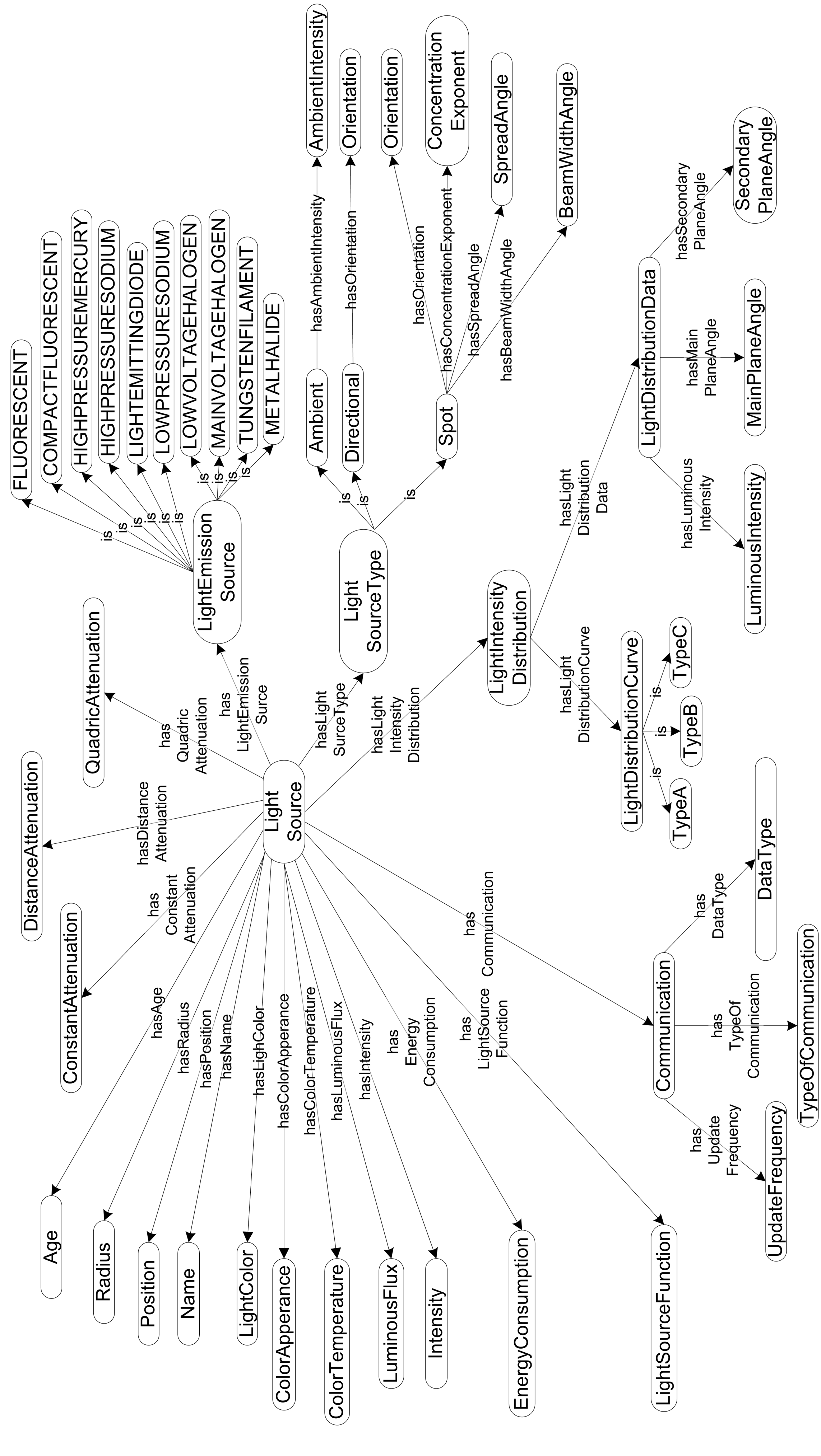


Figure 6: Lighting domain ontology.

The entities in the lighting ontology in Figure 6 are defined as follows:

- Lighting - Definition from IFC: The light source entity is determined by the reflectance specified in the surface style rendering. Lighting is applied on a surface by surface basis: no interactions between surfaces such as shadows or reflections are defined.
  - Name (type: Literal) - Definition from IFC: The name given to the light source in presentation.
  - Age (type: Numeric) - Number of hours (seconds) that a light has been used.
  - LightColor (type: RGB) - Definition from IFC: The color field specifies the spectral color properties of both the direct and ambient light emission as an RGB value.
  - AmbientIntensity (type: Real) - Definition from IFC: The ambientIntensity specifies the intensity of the ambient emission from the light. Light intensity may range from 0.0 (no light emission) to 1.0 (full intensity).
  - Intensity (type: Real) - Definition from IFC: The intensity field specifies the brightness of the direct emission from the ligth. Light intensity may range from 0.0 (no light emission) to 1.0 (full intensity).
  - EnergyConsumption (type: Numeric)
  - LightingFunction (type: Literal) - How lighting is used, e.g. ambience, environmental illumination etc.
- Light source ambient (Subclass of: Lighting) - Definition from IFC: Light source ambient lights a surface independent of the surface's orientation and position.
- Light source directional (Subclass of: Lighting) - The class definies direction of the lighting.
  - Orientation (type: Direction) - Definition from IFC: The direction of the light source.

- Light source geometric (Subclass of: Lighting) - Definition from IFC: The class defines a light source for which exact lighting data is available. It specifies the type of a light emitter, defines the position and orientation of a light distribution curve and the data concerning lamp and photometric information.
  - Position - Definition from IFC: The position of the light source. It is used to orientate the light distribution curves.
  - ColourAppearance (type: RGB) - Definition from IFC: Artificial light sources are classified in terms of their color appearance. To the human eye they all appear to be white; the difference can only be detected by direct comparison. Visual performance is not directly affected by differences in color appearance.
  - ColourTemperature (type: Real) - Definition from IFC: The color temperature of any source of radiation is defined as the temperature (in Kelvin) of a black-body or Planckian radiator whose radiation has the same chromaticity as the source of radiation. Often the values are only approximate color temperatures as the black-body radiator cannot emit radiation of every chromaticity value. The color temperatures of the commonest artificial light sources range from less than 3000K (warm white) to 4000K (intermediate) and over 5000K (daylight).
  - LuminousFlux (type: Real) - Definition from IFC: Luminous flux is a photometric measure of radiant flux, i.e. the volume of light emitted from a light source. Luminous flux is measured either for the interior as a whole or for a part of the interior (partial luminous flux for a solid angle). All other photometric parameters are derivatives of luminous flux. Luminous flux is measured in lumens (lm). The luminous flux is given as a nominal value for each lamp.
  - LightEmissionSource (type: Literal) - Definition from IFC: Identifies the types of light emitter from which the type required may be set. Possible values:
    * COMPACTFLUORESCENT,

    * FLUORESCENT,
    * HIGHPRESSUREMERCURY,
    * HIGHPRESSURESODIUM,
    * LIGHTEMITTINGDIODE,
    * LOWPRESSURESODIUM,
    * LOWVOLTAGEHALOGEN,
    * MAINVOLTAGEHALOGEN,
    * METALHALIDE,
    * TUNGSTENFILAMENT,
    * NOTDEFINED.

- Light source positional (Subclass of: Lighting) - Definition from IFC: Desribes lighting position and attenuation coefficients.
  - Position (type: Cartesian Point) - Definition from IFC: The Cartesian point indicates the position of the light source.
  - Radius (type: Numeric) - Definition from IFC: The maximum distance from the light source for a surface still to be illuminated.
  - ConstantAttenuation (type: Real) - Definition from IFC: This real indicates the value of the attenuation in the lighting equation that is constant.
  - DistanceAttenuation (type: Real) - Definition from IFC: This real indicates the value of the attenuation in the lighting equation that proportional to the distance from the light source.
  - QuadricAttenuation (type: Real) - Definition from IFC: This real indicates the value of the attenuation in the lighting equation that proportional to the square value of the distance from the light source.

  (type:)

- Communication (Subclass of: Lighting) (type:) - Description of communication tha lighting can have with other devices.

  - UpdateFrequency (type: Numeric) - The frequency of comunication between light and other devices.
  - TypeOfCommunication (type: Literal) - Type of communication between lighting and other devices.
  - DataType (type: Literal) - Type of data sent by light to other devices.

- Light distribution data source (Subclass of: Light source geometric) - Definition from IFC: The data source from which light distribution data is obtained.

- Light source spot (Subclass of: Light source positional) - Describes light source spot

  - Orientation (type: Direction) - Definition from IFC: The direction field specifies the direction vector of the light's central axis defined in the local coordinate system.
  - ConcentrationExponent (type: Real) - Definition from IFC: This real is the exponent on the cosine of the angle between the line that starts at the position of the spot light source and is in the direction of the orientation of the spot light source and a line that starts at the position of the spot light source and goes through a point on the surface being shaded.
  - SpreadAngle (type: Real) - Definition from IFC: This planar angle measure is the angle between the line that starts at the position of the spot light source and is in the direction of the spot light source and any line on the boundary of the cone of influence.
  - BeamWidthAngle (type: Real) - Definition from IFC: The beamWidth field specifies an inner solid angle in which the light source emits light at uniform full intensity. The light source's emission intensity drops off from the inner solid angle (beamWidthAngle) to the outer solid angle (spreadAngle).

- Light intensity distribution (Subclass of: Light distribution data source) - Definition from IFC: The class defines the the luminous intensity of a light source that changes according to the direction of the ray.

  - LightDistributionCurve (type: Literal) - Possible values:
    * TYPE_A,
    * TYPE_B,
    * TYPE_C,
    * NOTDEFINED

- Light distribution data (Subclass of: Light intensity distribution) - Definition from IFC: The class defines the luminous intensity of a light source given at a particular main plane angle.
  - LumniousIntensity (type: Real)
  - MainPlaneAngle (type: Real)
  - SecondaryPlaneAngle (type: Real)

The lighting ontology as presented in Figure 6 was used in both of Sofia [1] and this project and is presented here to give and indication of the complexity of description of something as common as lighting.

4.1.4 Application-specific ontology

An application-specific ontology is designed to represent concepts and relations between them in a particular application or architecture. This ontology is the source of the knowledge for a device about a specific architecture or an application. This ontology is described using protocols, message types and configurations needed to enable cooperation between devices in a space. These ontologies enables the device to function in a particular architecture. This project is mainly focusing on Application-specific ontologies and how to design them to enable a device to cooperate in a proposed architecture (see Chapter 5). An application-specific ontology was designed for the needs of this project, see recipe ontology in Section 7.3.1.

4.2 Knowledge Representation

The knowledge representation presented in this project is inspired and influenced by knowledge representation schemes in AI. There are many knowledge representation schemes, the most popular are: logic, procedural, network and structured representations.

4.2.1 Logic representations

Logic representations scheme uses expressions in formal logic to represent the knowledge [17]. This representation uses propositions with predicates to express facts. An example of rule expressed with its logic representation is as follows:

$\forall x \forall y : father(x, y) \vee mother(x, y) \rightarrow parent(x, y)$

The universal rule presented above states that for all values of x and y, if x is the father of y or the mother of y then x is parent of y. This rule also contains a fact $parent(x, y)$ that can be used for further reasoning. This knowledge can be implemented with use of the PROLOG programming language. The drawback of representing knowledge with logic representations is that it operates under a closed world assumption [52], therefore it assumes that all knowledge is expressed in the knowledgebase, and only with this assumption can it reason. The logic representation is declarative, it specifies what is known about the problem or domain, not what actions to take to solve the problem or how to use the knowledge.

4.2.2 Procedural representations

A procedural representation represent a knowledge as a set of instructions for solving a problem[17]. Procedural expressions use *if-then* statements to express procedure: if a given condition is met then a series of actions are performed. The database of the knowledge is called a working memory and it represent the whole knowledge of the system at a given moment [17]. Following a similar example from [17], let's consider an example of the system gathering data about a user and determining if it is an adult or under age. At a start the system's working memory is as follows:

<user surname? is unknown>

<user forename? is unknown>

<user age? is unknown>

<user adult? is unknown>

<LEGALAGE is 18>

Where user's surname, forename, age and state of being an adult is unknown and *LEGALAGE* is set to be *18.* The rules to find out more about the user can be as follows:

1. IF<user surname? is unknown>

   THEN ask ”what is your surname?”

   read SURNAME

   remove <user surname? is unknown>

   add <user surname? is SURNAME>

2. IF<user forename? is unknown>

   THEN ask ”what is your forename?”

   read FORENAME

   remove <user forename? is unknown>

   add <user forename? is FORENAME>

3. IF<user age? is unknown>

   THEN ask ”what is your age?”

   read AGE

   remove <user age? is unknown>

   add <user age? is AGE>

4. IF<user age? is AGE> and AGE≥LEGALAGE

   THEN remove <user adult? is unknown>

   add <user adult? is YES>

5. IF<user age? is AGE> and AGE<LEGALAGE

   THEN remove <user adult? is unknown>

   add <user adult? is NO>

Let's assume that the user is Tom Brown age of 23. Given the contents of the database, the following sequence of rules execution is expected to occur:

- Rule 1 is executed, the user types in *Brown* and the entry <user surname? is Brown> is added to the database.

- Rule 2 is executed, the user types in *Tom* and the entry <user forename? is Tom> is added to the database.

- Rule 3 is executed, the user types in *23* and the entry <user age? is 23> is added to the database.

- Rule 4 is executed since the user's age, equal to 23, is larger than the predefined value of LEGALAGE, equal to 18. This rule replaces entry <user adult? is unknown> with entry <user adult? is YES>in the database.

The interpreter examines the rule set every time from the beginning [17]. In more complicated systems some conflicts can occur. As rules triggers when the condition is met it can also lead to infinite loops.

4.2.3 Network representations

Another knowledge representation schemes in AI is network, that capture knowledge as a graph, where nodes are concepts and objects and arcs represent relationships between them [17]. This scheme can use ontology to represent concepts and relationships between them. Similar to ontology, network representations support property inheritance. If a concept or an object is a member of a class it inherits all attribute values from the parents, unless alternative values are overwritten [17].

The use of network representations and an ontology helps to structure and organise the knowledge. This representation is useful when the object or a concept is associated with many attributes and when relations between concepts are important [17]. The graph structure of the knowledge schema enables quick navigation between concepts and more human-like understanding of concepts and because of inheritance enables discovering similarities between concepts.

4.2.4 Structured representations

More complicated knowledge schemas, called structured representations use slots to represent attributes in which values can be placed [17]. The values that can be placed in slots is either an instance-specific information or a default, stereotypical value associated with a slot.There are two most common types of structured representations: frames and scripts.

Frames are designed to represent stereotypical information about an object [45]. An example of an object described by frames can be an office. Most of office spaces have many desks, chairs, shelves, documents in folders and books, computers, printers and phones. The information about an office can be stored in a network of frames, where frames have slots with particular values. A slot can be for example the number of desks in an office, that for a particular office can have value of 5. Scripts is next type of structured representations. Scripts are very similar to frames, it represents stereotypical situation [17]. Frames describe objects, scripts unlike frames represent situations. Scripts were first proposed to reason about contextual information to support natural language processing. As discussed in [17], scripts consists a number of elements:

- entry conditions - conditions that need to be true in order to run the script,
- results - facts that are true when a script finishes,
- props - items involved in the event,
- roles - expected actions to be performed in the event described by the script,
- scenes - sequence of events,
- tracks - various themes of patterns of the script.

Scripts are used when *common knowledge* needs to be used to interpret a sentence. Following an example from [17], let's consider a sentence:

Alison and Brian received two toasters at their engagement party, so they took one back to the shop.

To interpret this sentence the knowledge about presents, returning goods, engagement parties and toasters is needed. The common knowledge is that under normal circumstances people would't need to have two toasters, that's why its natural they want to return one. The situation would be totally different if the gift was two books. Therefore the rule that people do not want two similar objects only applies to some objects.

Scripts are useful when representing knowledge in a particular context and can be limited to a domain. This scheme provides an explicit and efficient mechanism to capture complex structured information within a context and a domain [17].

4.2.5 Chosen representations

The chosen knowledge representation is a mix of the presented representations. The inspiration and influence of AI theory is as follows:

1. Procedural representations - the structure of the system: working memory (see short-term knowledgebase in Section 7.4), set of rules (see recipes in Section 7.3) in *if-then* form and interpreter (see KBIE in Section 8).

2. Network representations - use of ontologies (see Section 4.1.2, 4.1.3 and 4.1.4) to structure the knowledge (see Sections 7.1, 7.3.1), use of inheritance when describing similar objects.

3. Structured representations - use of script-like description of a behaviour with entry conditions (recipe header see Section 7.3), scenes (recipe steps see Section 7.3, tracks (conditional choices see Section 7.3.2).

As presented by Finlay and Dix [17] the main knowledge representation schemes' requirements are: expressiveness, effectiveness, efficiency and explanation. An expressive scheme can handle different types and levels of granularity of the knowledge. It is able to represent specific and general knowledge. An effective scheme provides a means of inferring new knowledge from old. The third measure of a good representation scheme is efficiency. The scheme need not only support inference of new knowledge from old but also do so efficiently so the new knowledge can be used. Also the gathered knowledge must be usable and well represented. Finally a good knowledge representation has to provide an explanation of its inferences and allow it to justify its reasoning.

The chosen mix of presented schemes used in this project also follows the requirements of a knowledge representation in AI.

- expressiveness - The inspiration from procedural representations brings an advantage of procedural knowledge in situations when knowledge changes over time. This is useful when the initial and final state of the system differ between users. The network representations scheme, by use of an ontology, brings the ability to express knowledge with the use of facts and objects and relations between them. The hierarchy in a graph allows representing general and specific knowledge. The inspiration from structured representations, especially Scripts, allows making a description of a behaviour expressive in a particular context.

- effectiveness - The inspiration from procedural representations effects creation of new information in the knowledgebase. It is created by use of operators to change the contents of the working knowledge. Because of the use of the network representations, the inference is supported through property inheritance. With use of Scripts, the new knowledge can be deduced using context information, therefore the situation can be interpreted differently in different contexts.
- efficiency - The use of procedural representations make it transferable between different domains. The use of network representations reduces the size of a knowledgebase, because the properties can be inherited not explicitly attached to every class or entity. Consistency is well maintained with the inheritance between classes. The use of structured representations makes it possible to capture complicated knowledge in an efficient way.
- explanation - In the procedural representation a decision making process can be easily tracked by checking what rules have been run. In network representations the relationships and inference are explicit in the network links.

The hybrid created with use of AI theory of knowledge representation and new ideas presented in this project follows the recommended requirements. Therefore its feasible to create a knowledgebase based on this knowledge representation that can be used for reasoning.

5 Smart lighting project

Home automation systems are very popular nowadays. Although there are many home automation systems, there are still many unsolved issues and unattained targets. The project's goal is to create a system that customizes a lighting environment in a house or commercial environment in an easy and intuitive way. The objective is to make lighting systems more context-aware and able to adapt to changing environment and users' needs.

Most existing lighting automations consist of a central unit that controls the system and makes all the decisions. In this case light sources are only able to receive information about the controller's decisions. Therefore the controller is the main decision making component and responsible for managing context and users' needs. The central unit also has to store or has instant access to light settings and decide about configuration. Most of the controllers on the market have already been deployed with fixed functions and customizing them is usually impossible. For example The Living Colors lamp by Philips (*www.lighting.philips.com*) can produce any light color with use of 4 LEDs, but remote can control only one lamp at a time and choice of colors is fixed. A contemporary lighting system has to meet the requirements of the user and be dynamic and flexible enough to give their customers the desired performance. The technology is developing so fast that it is possible to put processing engines or even processors in lights and sensors, so they become able to make their own decisions and process data. With this extension in functionality, simple components can become more sophisticated. On the other hand, the engine that is deployed into lights or sensors should be as simple as possible and energy efficient, so the whole cost of the system consisting of installation, use and maintenance can be low.

5.1 Project specification

The smart lighting system is part of the Sofia (Smart Objects For Intelligent Applications) EU ARTEMIS funded project. Sofia is investigating the use of pervasive systems in the application areas of Personal Spaces, Smart Indoor Spaces and Smart

Cities (Figure 7). Integrative service components that cover User Interaction and Interfaces, Architecture and Application Development are also being undertaken.

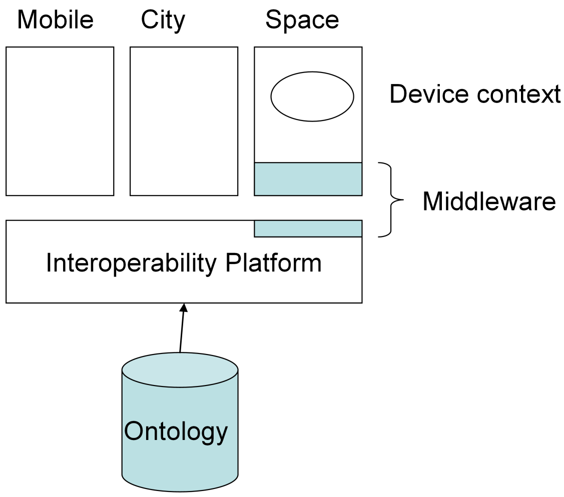


Figure 7: Sofia context.

One of the Sofia objectives is to create an interoperability platform to enable collaboration between multi-vendor devices, using different communication links, operating systems and software components. The interoperability is achieved using a common ontology and data format to exchange information about the smart space in a pervasive system. The interoperability not only enables cooperation between different devices but also between systems based on different architectures. The Sofia project uses open source platform Smart M3 [30] that enables information sharing between software entities and devices. This project follows the Sofia approach to use ontologies for interoperability and follows the Semantic Web of the World Wide Web Consortium (W3C *www.w3c.org*) approach for representing semantic knowledge with use of ontologies and a Resource Description Framework. The W3C is proposing RDF for a Semantic Web standard for data interchange [40]. The research reported in this thesis builds upon and improves an architecture called Smarties [62]. The extensions will be explained in the next sections. The research extends the Sofia approach by extending reasoning in devices by adding a knowledgebase and dedicated interpretation engine.

5.2 Basic architecture concepts

The architecture for a distributed system, called Smarties, is presented in [62]. In Smarties devices are autonomous and can connect in ad-hoc manner. Devices can organise themselves, execute some tasks without intervention of any central control component. In the extension to the basic Smarties architecture, knowledge of the system is distributed among devices that know what they have to do in a particular situation and perform a certain function. Therefore the scope of interest of the device is its co-operation with other devices. Every device is addressable, in this architecture numbers were used to identify devices in a space. A mechanism to assign an identification number to a device was out of scope of the presented architecture. Its possible that the identification is assigned by the communication mechanism, for example if devices communicate over TCP/IP, the identification can be an IP address.

5.3 Virtual devices and interactions

In the Smarties architecture to perform some task in an environment devices have to be selected and grouped into subnetworks. Therefore a subnetwork of devices in an intelligent system is called a *virtual device*, because the subnetwork acts like a distributed device to perform a task, see Figure 8.

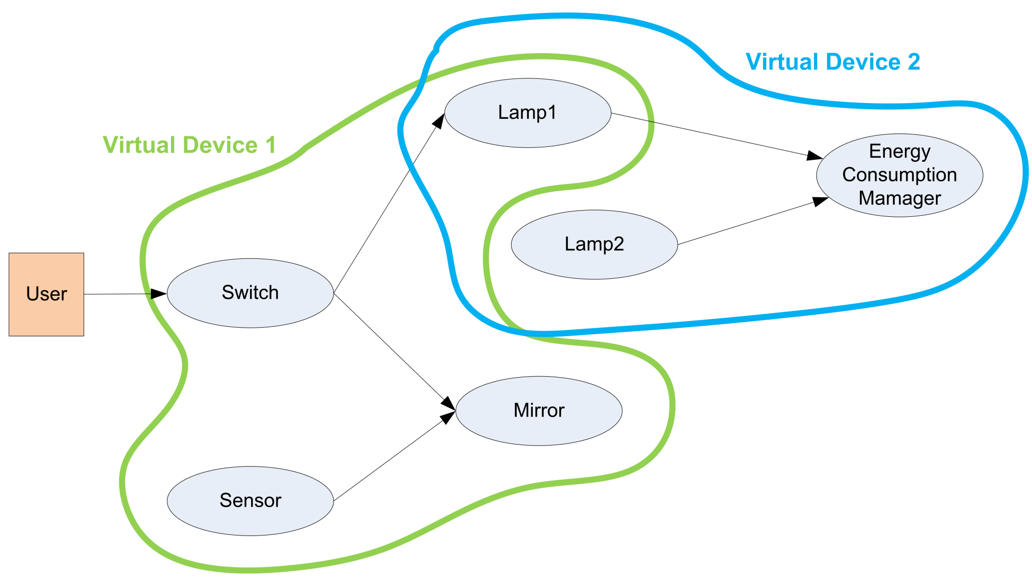


Figure 8: Different virtual devices in one environment. *Virtual Device 1* is composed because user requested to change lights in a single area of the room. *Virtual Device 2* was created to measure energy consumption used by lamps in this room.

A virtual device can be formed as a result of a user's request or some scheduled task that is being performed by a device. Some devices can participate in many virtual devices.

The device can be in any of four states (Figure 9): idle, execution and control, configuration, destroy. In the idle state the device is in an initial state and can accept requests for establishing new virtual devices. In the configuration state the device is waiting for other device responses or scheduled tasks to appear, in this state the device indicates that the configuration has a pending status.

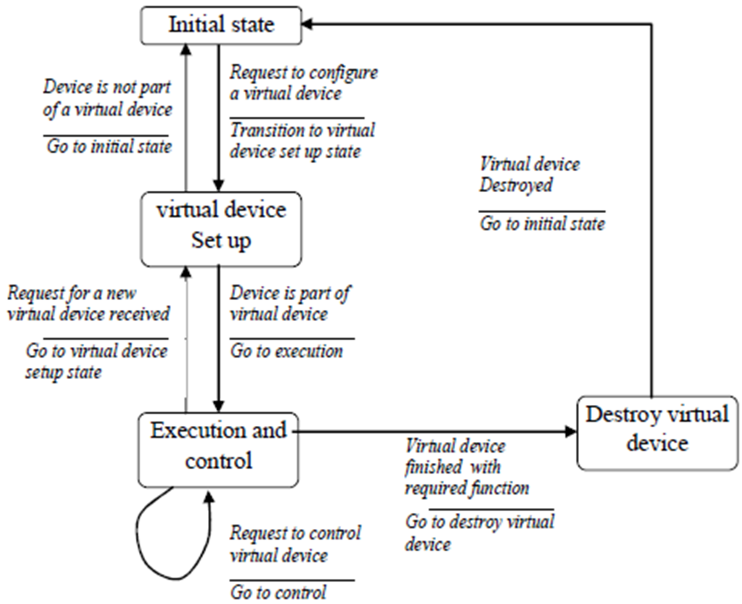


Figure 9: State diagram of the device [62].

In the execution and control state the virtual device has been established and devices work now as a distributed system that can receive control messages. In the destroy state the virtual device is destroyed and configuration data is deleted from device's memory. Devices that can participate in many virtual devices can be in many states at one time, therefore depending on a request the device is ready to behave according to any of the states.

5.4 Message format

Devices communicate only using messages. All messages sent in the system are broadcast messages, every device receives the message and decides if it should accept or discard it. In the original architecture [62] there are 4 types of messages, namely:

search, control, destroy and acknowledgement (Figure 10).

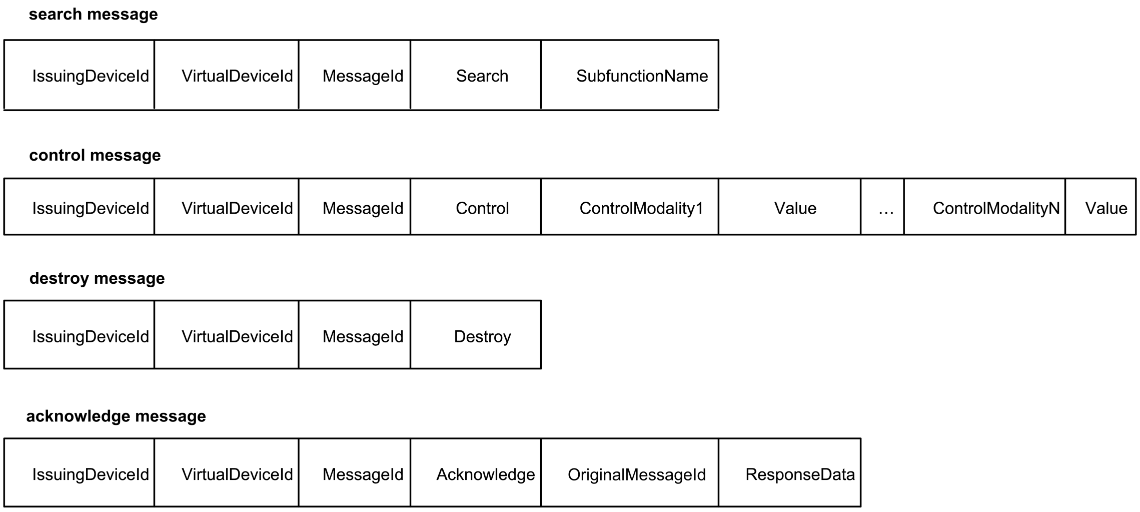


Figure 10: Message format from [62].

Messages in the improved architecture consist of header and payload. The header is always the same and consist of fields: *Issuing Device ID*, this field holds the device's unique identification number; *Virtual Device ID*, that is informing to what virtual device the message is issued or states the identification number of the virtual device that is about to be established; *Message ID*, that holds information about the message identification number. The header contains basic information about the message and also helps filtering out the messages that are not intended for a device. The payload part of the message is specific to each message. The first field of the payload is stating the type of the message.

The search message is used to search for a service when a new virtual device is needed. In this architecture its is only possible to search for a sub-function, that can be a service or a functionality offered by a device. To answer any request, an acknowledge message is used. The payload of the acknowledge message contains the *Original Message ID* that contains information about the message that the device is acknowledging, the *Response Data* field contains any information that the device is required to answer on a request. The destroy message is very simple and only indicates which virtual device needs to be destroyed. In order to influence the virtual device after it was established, the control message is used. The payload of this message contains a set of pairs of control modalities and values assigned to them. For example if the dim level needs to be changed to 50%, a pair $(dim, 50)$ will appear in the control message. In this the size of the set of pairs can be unlimited, in practice this could be constrained with the size of the message allowed to be sent over the communication link.

The message format proposed in [62] was extended with new message types and modification of message fields to improve existing capabilities and enable new functions of the system. In the proposed architecture there are ten message types as presented in Figure 11. All messages in the presented architecture consist of

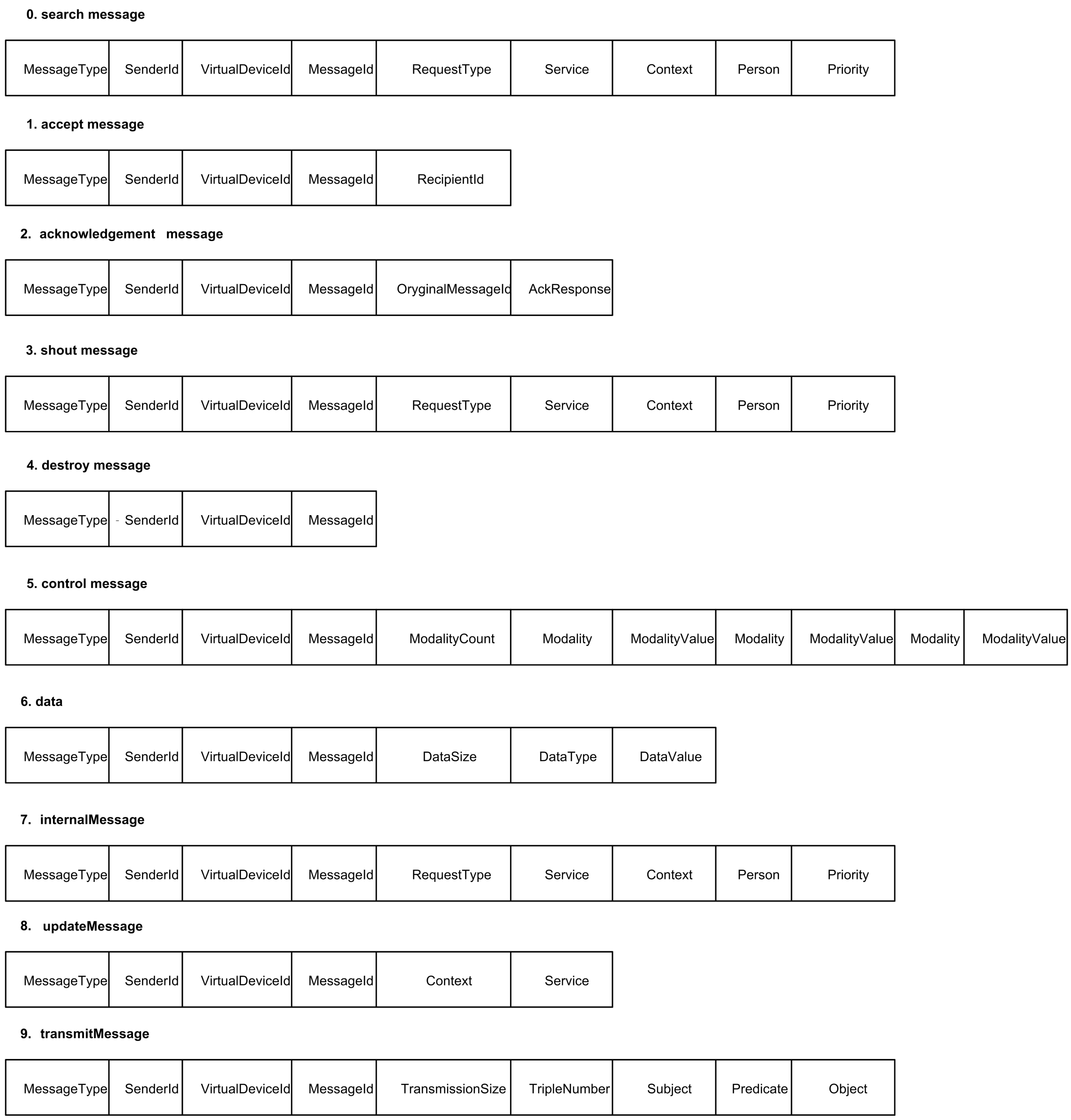


Figure 11: Message format.

header and payload sections. The proposed header is similar to the original one, extended only with the *Message Type* field. Fields in the header were rearranged, so the *Message Type* is the first field in the header. This was done because the size of the message changes depending on the type, to enable automated message parsing, the interpretation component only reads the first field of the message and matches it with message structures understood by the device. The exact mechanism of parsing messages is presented in Chapter 8, the protocol regulating a communication mechanism is presented in Section 5.5.

5.4.1 Search message

The search message is used to establish virtual devices. Fields *Request Type*,*Context*, *Person* and *Service* describe a type and purpose of the request. The *Priority* field indicates the importance of the message. A request to create a virtual device with a higher priority can overrule the existing virtual device configuration.

5.4.2 Accept message

The accept message is sent to inform any device that it is being accepted as a part of a virtual device. The field *RecipientId* contains an identification number of a specific device.

5.4.3 Acknowledge message

The acknowledge message is used to answer requests issued by devices. The *Original Message Id* field indicates what acknowledge message this message is responding to and the *Ack Response* field contains any information that the device is required to answer on a request. The acknowledge message is only accepted by a device that is expecting this kind of message, otherwise the message is dropped.

5.4.4 Shout message

The shout message is used to asses if the device can participate in a virtual device with requested capabilities. This message is very similar to the search message, but triggers a response from a device, rather than actions to participate in a virtual device. The shout message is used to asses a situation in the smart space, wherever there are devices participating in a virtual device and to determine their configuration state.

5.4.5 Destroy message

The destroy message uses the field *Virtual Device ID* to destroy a configuration associated with the requested virtual device. This message is needed while reconfiguring and to enable simple destruction of selected virtual devices.

5.4.6 Control message

Similar to the original Smarties architecture, in order to influence the virtual device after it was established, a control message is used. The payload of this message contains sets of pairs of control modalities and values assigned to them. There can be one to three sets of modalities in the control message, the number of sets is expressed using the *Modality Count* field. Modalities are commands that a device understands, these modalities are next mapped in the device to commands that are used to influence device functionality.

5.4.7 Data message

The data message is used to propagate data in the system. The information carried by this message consist of pairs of $(Type, Value)$ represented in message fields *DataType* and *DataValue*. The number of pairs can vary and and is contained in a field *DataSize*. The data message is issued only for one virtual device, and the device itself needs to know how to interpret the data.

5.4.8 Internal message

The internal message is used to trigger internal actions in the reasoning component. This message is used to find an appropriate action within a device and run a set of instructions from the knowledgebase.

5.4.9 Update message

The update message is usually sent by a new device in a space and is used to assets whether other devices provide a specific *Service* and understand its *Context*. The update message is used if the receiving device needs an update, if so an updating mechanism is triggered.

5.4.10 Transmit message

The transmit message is used to transmit triples of data: *subject, Object* and *Predicate*, explained later in Section 7.1.1. This message is a part of a transmission mechanism, the number of messages to be sent (size of data to be transmitted) is indicated by the field *TransmissionSize*. The field *TripleNumber* contains the sequence number of the data being sent. If one or many of messages have been lost, messages with a particular triple number can be retransmitted.

5.5 Protocol

The protocol to establish a virtual device is a four-way handshake protocol, similar to three-way handshake used in the TCP protocol [10]. To start the mechanism a request is needed, it can be issued by a user, other device or the device itself. An example of the use of the protocol is presented in Figure 12. To establish a virtual

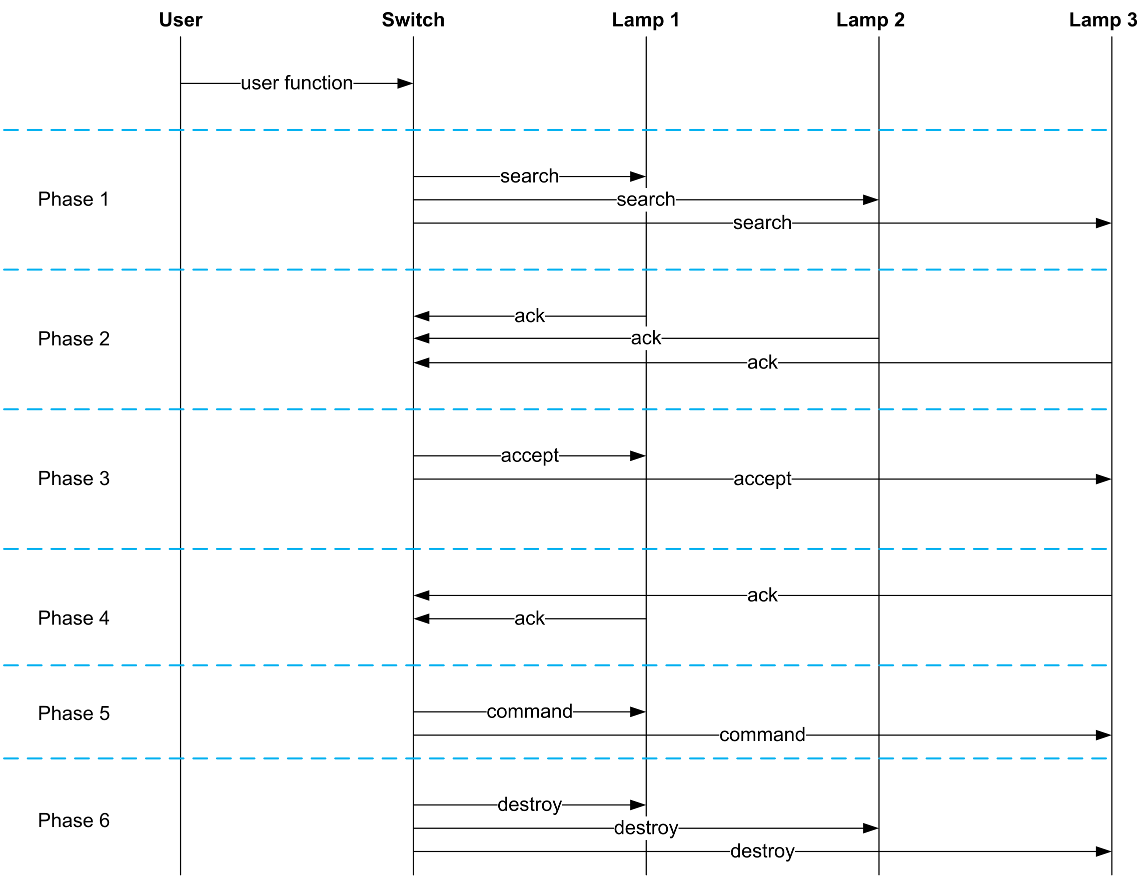


Figure 12: Four-way handshake protocol.

device a search message needs to be sent (Figure 12, phase 1). This message is sent to all the devices in order to asses which devices posses the requested capabilities and information. If the device is ready to establish a new virtual device it sends an acknowledgement message to the virtual device members (Figure 12, phase 2). Lamps now start a new configuration and remember data associated with the virtual device. This configuration is still in the pending state and if a lamp does not get the expected messages on time, the configuration will be deleted. In phase 2 only the Switch accepts the acknowledging messages and decides which devices can be accepted into the new virtual device. In phase 3 accept messages are being sent only to selected devices, in this case lamp 2 was not accepted, therefore it deletes the configuration and it does not participate in the virtual device. Lamps 1 and 3 continue the protocol in phase 3 by sending acknowledge messages and change the configuration state to engaged. From this moment devices lamp 1, lamp 3 and switch belong to one virtual device. To influence devices a control message can be sent (Figure 12, phase 5). To destroy an established virtual device a destroy message can be sent (Figure 12, phase 6), in this case all devices associated with the virtual device delete their internal configuration and wait for further requests.

In the original architecture in Smarties the protocol to establish a virtual device was a two way handshake, therefore the requesting device needed to accept all the devices that answered the request. The presented protocol is designed as a four-way handshake to ensure that both parts agree to participate in the virtual device and to enable the issuing device to choose participants, the decision can be based on their location, capabilities or any other information that is sent in the acknowledging message. The four-way handshake is a more reliable protocol, therefore both sides of participants are sure that the virtual device has been established.

5.6 System components

The system consists on many autonomous devices. All device make decisions among themselves, so the intelligence is embedded in every device, as it is a specialist in its own domain, therefore no central control or repository is needed. The decision to distribute the decision-making process influences the processing and reasoning capabilities in devices. Therefore devices need knowledge and reasoning capabilities on top of device functionality that is already in the device. This project investigates the design of a generic interpretation engine for reasoning and representing knowledge in a machine readable form.

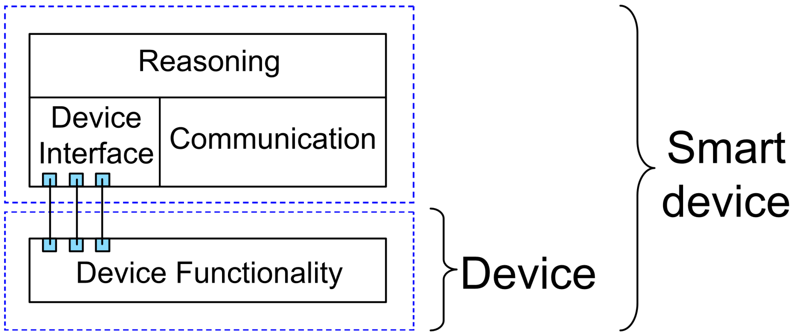


Figure 13: Device architecture.

In order to communicate between the decision making mechanism and device functionality, a device interface is needed; it is also necessary for a device to co-operate with other devices, therefore a communication component is needed. The separation of concerns is visible in Figure 13, the device can stay unchanged, the additional component enabling reasoning needs only an interface to the device and knowledge about a device's capabilities and offered services. The reasoning component is able to make decisions and use the communication link to send messages and use the device interface to change the device's behaviour. The reasoning component is further discussed in Chapter 8.

5.7 Knowledgebase outline structure

The knowledgebase represents a set of information the device needs to know and can be divided into four main parts: information about the device (type, domain), functions and services offered by the device, information about the environment, actions and behaviour the device can perform.

All this information is needed for the device to make a decision how to behave in a particular situation. Messages and the protocol need to be described in the knowledgebase in order to know what rules to communicate in a specific architecture like Smarties configuration protocol. The knowledgebase construction and its interpretation is the key to enable the whole system to behave in a particular way and are discussed in Chapter 7. The capabilities of the architecture and the presented system with case studies based on a simple home automation example are presented in Chapter 6.

6 Case Studies

When considering home, office or warehouse spaces there are many types of systems supporting them, for example: security, heating, ventilation and air conditioning, lighting and audio-video capability. Contemporary building automation systems are targeted to reduce energy use and meeting users' preferences.

Lighting systems at home or in the office environment consist of many light sources, switches and dimmers in many rooms. The control system has to take into consideration the variety of lighting equipment, location in the space, mobility of light sources, dynamic number of devices, changing space configuration, users' preferences and environmental influences. Contemporary lighting systems need to be intelligent, flexible and adapt to users' needs and offer energy consumption management capabilities. The idea is to use concurrent programming techniques to implement the infrastructure of a lighting system consisting of many devices with different capabilities.

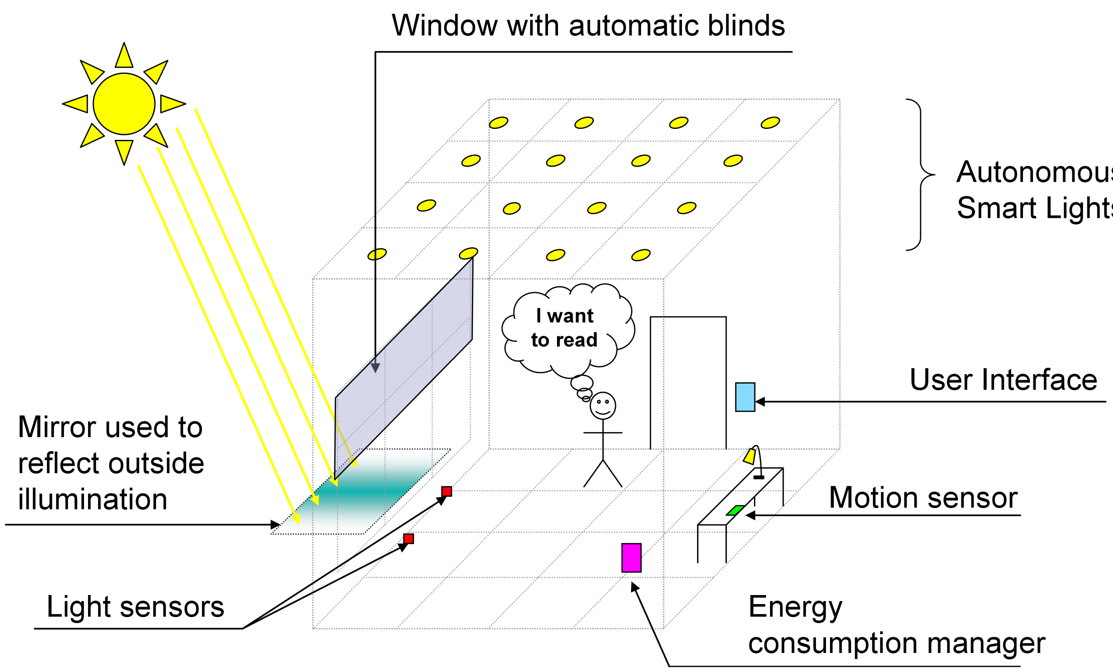


Figure 14: Proposed environment.

The environment chosen to describe the system is presented on Figure 14. Figure 14 shows a model of a room which is used as an office. The space is equipped with a light switch with a user interface (UI), ceiling lamps, lamp on a desk, automatic blinds on the window,a mirror used to reflect daylight to illuminate the room, light and motion sensors. The user is the center of attention and can set his/her preferences manually in the UI. When the user indicates the need for more light, the switch performs a search for devices that provide the service lighting for this user. All the lamps, blinds and mirror respond to this request, because all of them are able to provide more light for the space.

The other task that the set-up can perform is to keep the light at the same level in whole room. This configuration is using communication between light and sensors to use as much natural light as it is possible, while attaining a stable light level.

6.1 Context

There are many definitions of context, in this project we consider a context to be a user intent or wish, or environmental conditions (see Section 2.5). Therefore context can be associated with environment conditions, presence of particular devices or people, users' intent, or periods and events associated with a calendar. Devices in the architecture react differently to different contexts, all the devices usually react in an autonomous way and perform actions that are associated with a particular context. For example the context *SunnyDay* would trigger different behaviours in a lamp and a radiator.

6.2 Proposed Scenarios

In this section three scenarios are described that exploit different capabilities of the case study environment. The proposed architecture (see Section 5.3) enables devices to organise into virtual devices, and the protocol (see Section 5.5) enables sending control messages and data.

6.2.1 Scenario One

The first scenario is organising devices that perform one service, lighting, in one virtual device and control color and dimming level. When a user enters the room he/she chooses the context *Reading* on a panel on the wall. The switch initiates a new virtual device by sending a message to all the devices. Lamps and other light sources that understand the context *Reading* join the virtual device, every light source has settings associated with a particular context, for example one lamp changes color to red and dims to 80%. When the user wishes to change light source settings, he/she simply chooses the desired light level or color and a control message is sent to all devices participating in a virtual device. A user can change context by pressing a button on a switch panel, the previous virtual device will be destroyed, and a new configuration will be set with all the devices that understand the new context.

6.2.2 Scenario Two

The second scenario is using a context from the environment and scheduled tasks saved in a device. In this example when a user initiates the context *Reading*; lamps join the virtual device and also search for an indoor light sensor to adjust the light level according to its value. A mirror present in the space searches for indoor sensors to adjust dimming level and also an outdoor sensor to asses how much light it can offer the space. Therefore the mirror establishes two virtual devices with sensors in order to participate in the virtual device for context *Reading*. The desired light level is set by the user and associated with the context. In this scenario the light adjusting can be regulated by an algorithm embedded in devices in order to achieve emergent behaviour. The algorithm, presented in Chapter 9, helps to save energy while keeping the light in a space at a desired level.

6.2.3 Scenario Three

The third scenario demonstrates the learning capabilities of the system. In this scenario we describe how the system deals with forward-compatibility, therefore how to introduce devices and contexts that are not known in a space. If a projector enters the space with a new context *Projecting*, devices present in the space cannot participate in this context, therefore the projector has to asses if there are devices that can cooperate with it, check if the context is present in the space, and if necessary teach selected devices about the context *Projecting*. The next time the projector enters the same space, the initial configuration will not be needed. In this scenario, the projector carries a list of actions for all possible devices providing lighting services. The learning mechanism is very simple and relies on copying required actions and distributing them to selected devices. These action sequences are described in Section 7.3.

6.2.4 Tasks performed by the system

The scenarios were carefully selected to show the different possibilities of the described system.

The first scenario is showing the system can be used as a replacement for the existing lighting systems, all the basic capabilities are kept, the underlying architecture is changed from centralised to distributed. The self-configuration capability is shown here by establishing virtual devices for a particular situation performed automatically.

The second scenario shows how the system deals with context information. It also shows how the system can co-operate with sensors or deal with data from other devices. An algorithm triggering an emergent behaviour will be demonstrated to show an example of a complex adaptive behaviour that can be implemented in the proposed architecture.

The third scenario, a simple learning technique is presented to show how the system deals with re-programming, forward-compatibility and learning. This scenario shows the ease of installation and maintenance of the proposed system.

7 Knowledgebase Development

A knowledgebase is a database of any kind containing a knowledge represented in a machine readable format. A knowledgebase for small devices with simple functionalities is presented in this chapter. First in Section 7.1 a layered structure and a specific data format of the knowledgebase are described and explained. Next in Section 7.2 the knowledgebase content is presented and discussed on the example of a lamp. Section 7.3 presents a chosen format of representing devices behaviour in this specific knowledgebase format. Finally in Section 7.4 division of the knowledgebase is explained. The knowledgebase and the reasoning mechanisms are the centre of intelligence in an AI system, the knowledge needs to be well structured and represented according to rules so that if the knowledge is modified or updated, reasoning mechanisms can still make valid decisions.

7.1 Knowledgebase structure

The structure of the presented knowledgebase is based on an ontology. The ontology can offer a flexible model to structure and reuse data, it also enables semantic interoperability when exchanging knowledge between components. Neches [46] points out that knowledgebase should be constructed from many levels of knowledge. These layers are increasingly specialized, therefore layer division begins with very general concepts, based on an ontology model and finishes with very specialized and less reusable knowledge [46]. A branch of ontology that is used for constructing knowledgebase should consist of a general core or foundation ontology together with domain-specific and device-specific concepts and relations. The knowledgebase in a device or component should consist only of necessary concepts, depending on a component's functions and domain as described in [62]. The knowledgebase can be extended if new functionalities or services appear. The most specialized part of a knowledgebase are instances and the data associated with a particular device or component.

A knowledgebase consists of layers of knowledge (Figure 15). Layers are divided into T-Box and A-Box. T-Box consists of ontology models, like DOLCE [42] or any other upper level ontology, A-Box contains instances corresponding classes and properties from T-Box.

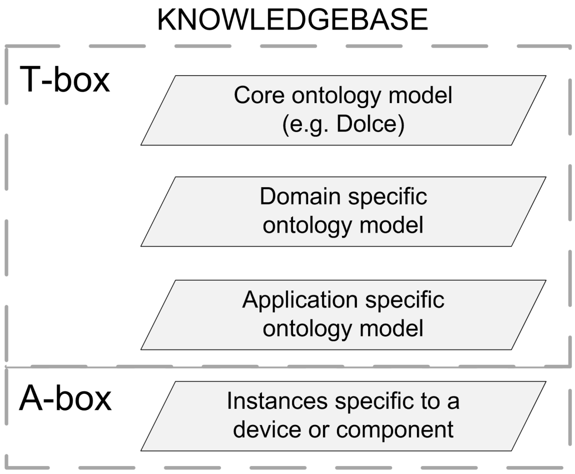


Figure 15: Layered structure of a knowledgebase.

The top level of the knowledgebase is a meta model of the core ontology. The meta model of the ontology has to be fixed, this project is following the Sofia [1] approach by choosing Dolce as its core ontology. Dolce is an upper ontology that addresses very general domains [42]. The next layer in Figure 15 shows a domain ontology that is created for a specific area. For example if the device is a sensor, its domain ontology describes classes and properties that are associated with the sensor domain. In this way the knowledge of the surrounding world is narrowed to the domain the device or component belongs to. The next layer, Application specific ontology model, is a model for instances that describe devices' behaviour, capabilities and services that the devices can provide and also rules of the architecture that the device belongs to. The last layer of the knowledgebase, categorized as an A-Box, contains instances describing a particular device according to rules provided by the T-Box ontology models. Similar devices will have a similar T-Box part of the knowledgebase, but different instances in A-Box. T-Box can be automatically generated and used when building the A-Box for a particular device.

7.1.1 Resource Description Framework

The models and the knowledge need to be represented in a machine readable form. As the knowledgebase follows an ontology model, it is also expressed using a data model recommended for describing ontologies. The structure of the data representation needs to be kept simple to enable ease of reading, modification and interpretation. The ontology concepts are widely used for interpreting semantic data and web resources in the World Wide Web Consortium (W3C) Semantic Web. The W3C approach uses languages based on Resource Description Framework (RDF) [40] like DAML+OIL [28], that is combination of DARPA Agent Markup Language founded by US Defence Advanced Research Projects - DARPA and OIL, that is Ontology Interchange Language; or its successor OWL (Web Ontology language) [59]. The RDF is designed to represent information in a simple but flexible way [40]. The RDF is used in the W3C Semantic Web project to represent data for applications that use open information models. The RDF also offers a structure for machine readable information for applications that process data that have not been created in the same environment as the application, therefore enabling semantic interoperability of the transported data [40].

This project uses the RDF standard to construct a knowledgebase.The RDF has an abstract structure representing a simple graph-based data model [40]. An RDF triple consists of subject, predicate and object, where subject and object are a class or an RDF URL reference to a class and predicate is a relationship between classes or a RDF URL reference to a relationship.

$$(Subject, Predicate, Object)$$

For example:

$$(apple\_tree, is-a, tree)$$

This triple uses predicate *is-a* to associate subject *apple_tree* with object *tree*. Ontologies are designed by domain experts and represent universal and domain-specific truths.

For the purpose of the World Wide Web ontologies are represented by the use of markup scheme like XML, for example when using OWL [59]. Unfortunately XML adds complexity to the database format by introducing tags that need to be interpreted. XML is not designed for data storage or efficient retrieval of data [36] the additional time to process data exists due to parsing and text conversion [6].

For devices with a limited processing capability it is important to keep the database structure as simple as possible. Therefore the knowledgebase format is kept to a simple three field RDF structure.

RDF for the World Wide Web can use many external files of different ontologies, therefore it is not essential to define all concepts in one RDF file. Devices from a pervasive system usually have a well defined functionality and often do not need to connect to the Internet. In many cases referencing by means of a URL to external ontology files is not possible. Therefore entries in the knowledgebase need only reference other entries in it.

7.2 Knowledgebase content

The knowledgebase consists of data necessary for the system to make decisions, process requests, interpret messages, learn and react to commands.

7.2.1 Smart Lamp

A lamp can be an example of a small device that can be part of the example system. The smart lamp consists of two parts (Figure 16). The first part is designed by a lamp manufacturer and consists of hardware, software and capabilities to turn on, off and dim. The second part is responsible for making a simple lamp a smart lamp. This part is processing context information and consists of a knowledgebase (KB) and a reasoning component connected to a smart lamp by the device interface.

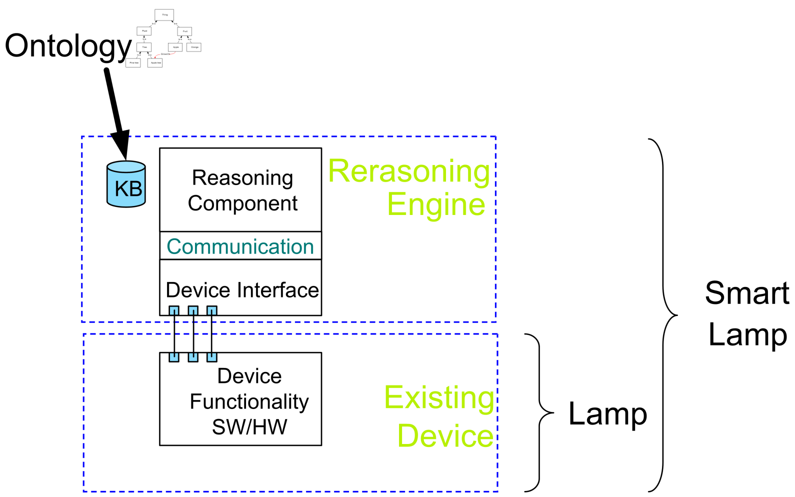


Figure 16: Smart lamp.

The knowledgebase is designed especially for this type of lamp, so it already has information provided by the manufacturer about the lamp capabilities. The reasoning component has access to the lamp's capabilities and can control it. The knowledgebase helps the device interpret situations in the environment or external commands from controllers and perform appropriate actions.

Following Neches approach [46], the lamp's knowledgebase is divided into increasingly specialised layers of knowledge(Figure 17).

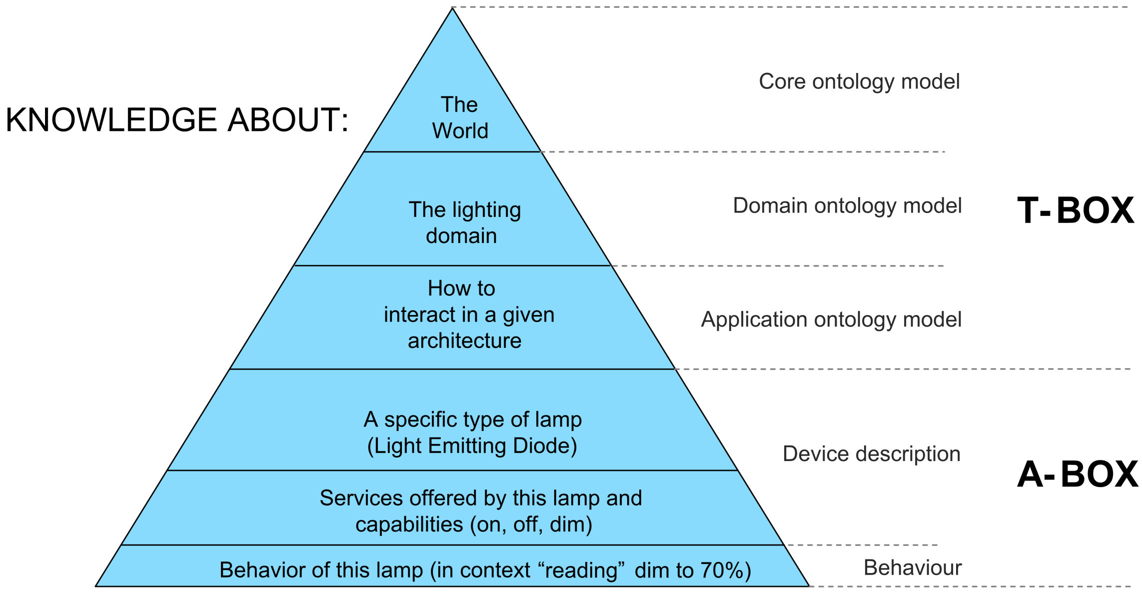


Figure 17: Lamp's knowledge pyramid.

The lamp needs to have only some general knowledge about the world, this knowledge comes from a foundational ontology like DOLCE [42] and helps the device understanding general concepts. The next layer is a knowledge about the lighting domain, this information needs to be more detailed, as a device needs to be specialized in its own domain. The last layer in T-Box is an application ontology model, where the device holds models of how it interacts in a given architecture. This layer covers information about protocol, architecture details, message format and general information associated with architectures in which the device can participate. Models from all of these layers help the device to understand the environment, its own domain and architecture in which it is participating. The next three layers from Figure 17 contain device descriptions and behaviour, these layers are specific for a particular lamp. The top layer of A-Box describes lamp's details, its type, location, name and other information from the lighting ontology (Figure 6 in Section 4.1.3).

The next layer of knowledge describes services offered by this lamp, for example *lighting*, and capabilities that the physical part of the lamp can offer, for example *on, off* or *dim*. This description is only valid for a particular lamp and it is usually provided by the lamp manufacturer. The last layer and the biggest in size describes the lamp's behaviour that is associated with architecture, protocols, context and environmental conditions.

The knowledgebase for any device consists of seven parts: device description, capabilities,context and users, recipes (see Section 7.3), configuration and state, and ontology models.

Device description. Description of the device using ontology. This part of a knowledgebase consists of two parts: a core ontology necessary to characterize the device universally, and a domain specific ontology describing entities and relations that are restricted to one type of device (for example: lighting, sensors, multimedia devices).

Device capabilities. This part of the knowledgebase contains descriptions of device capabilities that are provided by the device manufacturer. For example, capabilities are simple tasks that a device can physically perform. A simple lamp can only turn on, turn off and dim to some level. It is impossible for lamp to play music or accept coins because these tasks are beyond a lamp's physical capabilities.

Context and users. The context describes the actual situation of the whole system, or part of it. Every device can understand contexts that are described in its own knowledgebase. Contexts might be for example: *night*, *lunch_time*, *reading* or *watching_tv* and are associated with environmental conditions or actions performed by users. Understanding context and acting upon it is one of the most important tasks of the proposed system. Context is a very broad concept and customization for different users is important. For example *watching_tv* for *Anna* means that their personal phone should be off, window blinds shut and light dimmed. The same context can mean something different to *Tomas*, he wants his phone in loud mode, lights turned off, blinds shut. Devices participating in one context can be the same but settings might be different. Reasons for this are different preferences and priorities for particular users. Therefore the context *watching_tv* for *Anna* in a lamp or in a TV specify totally different actions.

Configuration and state. The information about the state and configuration of a device is very important and should be accessible by the device. For example devices to work as a group need to be formed into a virtual device [62]. The configuration holds information in which virtual devices the device participates and the current state of the configuration. A device that is a part of a virtual device should keep on functioning without any breaks, if a new request arrives a device has to be able to determine if it should drop its current task and respond to the request, by checking its current configuration settings. The state of a device is also included in the knowledgebase. For example a lamp can store information about its dim level or color in the knowledgebase where it can be easily accessed.

Recipes. Recipes are used to describe a procedure for the behavior of every device. Recipes are very detailed and specify the procedure that a device has to follow for a given context. The idea is to describe all important steps in a recipe, so the device only has to read the recipe and act upon it. Recipes are associated with context, people and request types.

Ontology models. This part of the knowledgebase consist of all models that a device needs to navigate in the ontology tree and state all relations between entities that are relevant for a particular device. Ontology models keep the knowledgebase structure and enables the reasoning mechanism to find data necessary to perform actions.

The knowledgebase is the centre of the intelligence of the system, all necessary information and behaviours are contained in this database. A knowledgebase is designed for particular devices, parts of T-Box can be similar for similar devices. The main part of the knowledgebase are recipes describing a device's behaviour.

7.3 Recipes

Recipes in a knowledgebase are sequential and represent steps that a device should follow in a particular situation. Every recipe has a header to describe the recipe's type, context and other conditions that are guarding access to this recipe.

The recipe shown in Figure 18 consists of n steps. Steps contain actions that

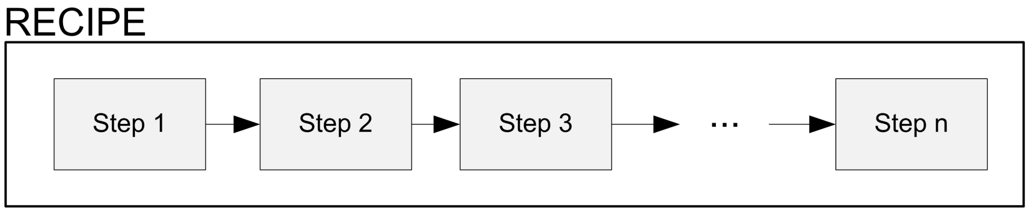


Figure 18: Recipe structure.

need to be performed in this particular step. Every step points to the step that needs to be executed next. The last step points to a special step, called *stepStop*,indicating the end of the recipe. If the device fails to finish a step, the recipe is dropped and the configuration data is erased. When a step requires some time constrains, the timer is started and the recipe is suspended, so kept in the same step, until the alarm is triggered. Once the time is up the recipe continues to the next step. In case the recipe reaches step *stepStop* the configuration status is set to *engaged*.

7.3.1 Recipes construction

Recipes are described using ontology terms, so the concept of a recipe has to be present in the ontology model. Classes and properties associated with the structure of a recipe are described in the T-box part of the knowledgebase in the Application specific ontology model layers described in Section 7.1 in Figure 15.

A simple example of an ontological model to represent recipes combined with DOLCE and Sofia models is presented in Figure 19. This model represents recipes and steps as a subclass of Dolce processes. Property *hasStart* associates the recipe with a first step for this recipe, and property *hasNext* creates a sequence of steps in a recipe.

Consider a lamp and a switch that can co-operate when grouped in one virtual device; the mechanism to group and configure these devices requires one device to start a configuration by sending a search message. When a device receives a search message it checks if it can participate in a virtual device that the message propagates and if it has a recipe that it can use in this particular configuration. If so, the recipe is executed, information about the configuration is saved; the device sends an acknowledgement massage back and acts upon the instructions provided by this recipe. If a switch initiates a virtual device with a lamp, the lamp has to have

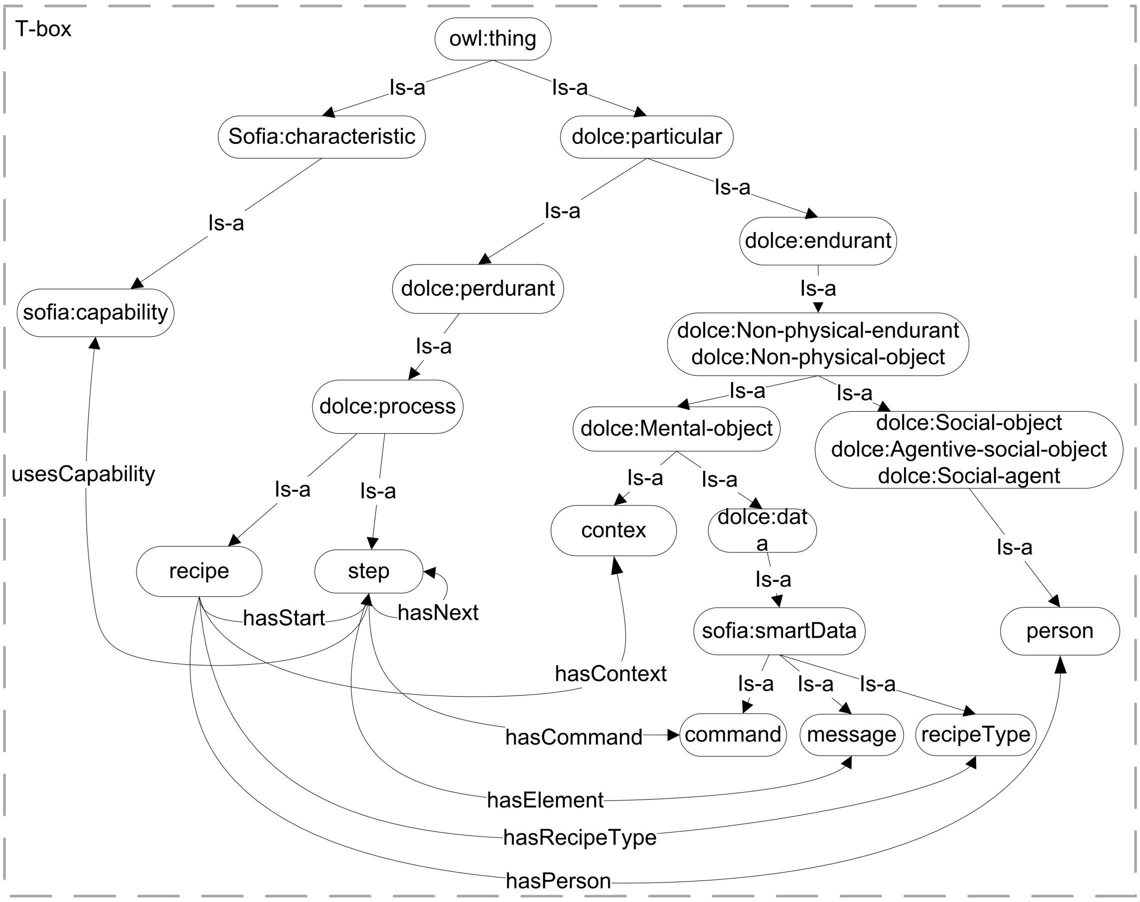


Figure 19: An ontological view of the concept of a recipe in T-box.

a recipe that describes, in detail, how to act upon a particular request. A recipe consists of header, that describes a recipe and steps the device has to perform. The header consists of recipe type, service and context of the recipe. A recipe can be associated with a person and customized for this person.

An example of a recipe for a lamp expressed in RDF format is shown in Figure 20. The *recipe1* consists of a header and four steps. Steps in a recipe represent a sequence of actions that the device has to perform. If a device reaches the end of a recipe, the task is finished.

A recipe represented in Figure 20 is described using RDF tripples. The same recipe can be illustrated with a tree (Figure 21). This tree shows relations between instances in the ontology and the corresponding model from T-box of the knowledgebase in Figure 19. Recipes are interpreted by a device with the use of a model from T-box (Figure 19) when searching for instances in A-Box (Figure 21). More complicated recipes are presented in Appendix A.

subject	predicate	object
recipe1	is-a	recipe
recipe1	hasContext	Reading
recipe1	hasPerson	Anna
recipe1	hasRecipeType	configureVD
recipe1	hasStart	step1
step1	is-a	step
step1	usesCapability	startConfiguration
step1	hasNext	step2
step2	is-a	step
step2	usesCapability	output
message1	is-a	message
step2	hasElement	message1
message1	hasMessageType	2
message1	hasAckResponse	0
step2	hasNext	step3
step3	is-a	step
step3	usesCapability	deviceFunctionality
step3	hasCommand	turnOnOff
step3	hasValue	1
step3	hasNext	step4
step4	is-a	step
step4	usesCapability	deviceFunctionality
step4	hasCommand	dim
step4	hasValue	70
step4	hasNext	stepStop
stepStop	is-a	step

Recipe's **header**. This recipe is associated with context reading and person Anna.

Step 1 saves all data needed for a particular configuration, therefore creates entries in local memory about the virtual device that the lamp is about to join.

Step 2 uses capability output to send an acknowledgment to a device that requested joining a new virtual device, in this case this message is sent to light switch.

Step 3 sends command to turn on the light.

In **step 4** lamp dims to 70 percent. A step called null indicates end of recipe. If a device reachEs the end of a recipe, a task is finished.

Figure 20: Example of a recipe in knowledgebase, expressed in RDF triples.

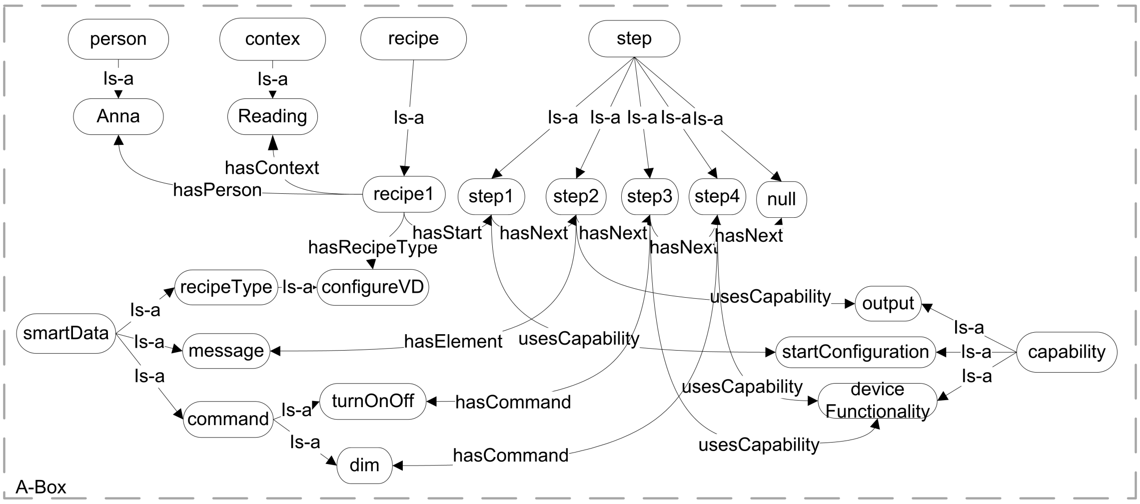


Figure 21: An ontological view of the concept of recipe1 from Figure 20.

7.3.2 Conditional choices

Recipes are used to inform a device the exact steps it should undertake in a particular context or when receiving data from different devices. Since devices may have many recipes , a choice on which recipe to run is made by the device using the content of the received message. If there is more than one recipe that are usable in a certain situation, then the message content is used to further refine a recipe choice. This is done by matching RDF triples of the recipe to fields of the message. This makes it possible to make conditional choices on which recipe to run and thereby achieve different device behaviour. A *Skip* mechanism was defined to enable *if and else* choices inside of the recipe. An example of a recipe flow is presented in Figure 22.

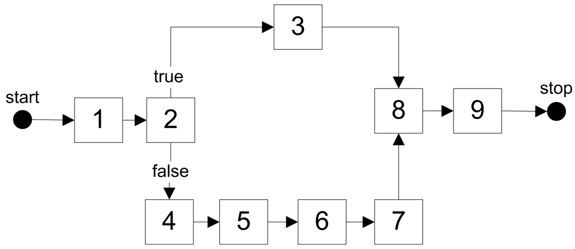


Figure 22: Skipping steps in recipe.

The *step 2* is constructed to evaluate a boolean value, prepared by *step 1*; if the value is true it directs the reasoning mechanism to *step 3* otherwise it skips *step 3* and continues to *step 4*. Both *step 3* and *step 7* point to *step 8*, so the recipe can be finished once *step 9* is performed. This way it is possible to alternate a behaviour inside of the recipe depending on a situation. More examples of recipes to analyse data using the skip mechanism are shown in Appendix B.

7.3.3 Self-configuration

A new configuration is usually triggered by an external request. It is also possible that the device recognises a need to start a new configuration with other devices. Therefore establishing one virtual device can trigger establishment of more virtual devices. For example, in Scenario Two in Section 6.2.2, mirror is requested to participate in a virtual device that regulates light intensity in a room. When reacting to a request, the mirror recognises that it needs an outside sensor to asses the light level in the external environment. The mirror needs to start a new virtual device with a light sensor and only then is it ready to participate in the virtual device regulating light intensity. The need to trigger a request for a new virtual device can be undertaken in parallel with other virtual devices. The mirror sends a special type of message, an internal message (described in Section 5.4.8). This message triggers running another recipe that configures a new virtual device.

The recipe that runs another recipe can wait for results from the other recipe or continue without acknowledgement. The mechanism to trigger running recipes internally, which is enabling self-configuration, is further described in Section 8.4.

7.3.4 Complex predicates

The predicates used in knowledgebase are very specific, to represent a relationship between two classes, a predicate with appropriate name is needed, for example to associate classes to class *Person*, predicate *hasPerson* is needed, these predicates are referred to as complex predicates. Complex predicates used in in the knowledgebase include, non-exhaustively:

acceptsMsgType	canConnect	hasAckResponse
hasAckResponse	hasAttribute	hasCapability
hasCommand	hasContext	hasData
hasDataSize	hasDataType	hasDataValue
hasField	hasFunction	hasId
hasMessageId	hasMessageType	hasModality
hasModalityCount	hasModalityValue	hasMsgIndex
hasNext	hasOperant	hasOrderId
hasOriginalMessageId	hasPerson	hasRecipe
hasRecipeType	hasResult	hasRule
hasSenderId	hasService	hasStart
hasTimer	hasValue	hasVirtualDeviceId
is-a	usesCapability	

The number of predicates grows when a new class or a new relationship type is added. The main advantage of complex predicates, from the point of view of constructing the ontology, is that it is possible to create different relations between the same classes. Unfortunately this poses problems when interpreting predicates by the reasoning mechanism, because the mechanism can interpret only a fixed number of predicates. If the predicate list is being extended, the reasoning mechanism might not understand new predicates. Let's consider an example where the knowledgebase contains triples:

(Person, is-mother-of, Person)

(Person, is-sister-of, Person)

(Teresa, is-a, Person)

(Anna, is-a, Person)

(Ela, is-a, Person)

(Teresa, is-mother-of, Anna)

(Teresa, is-sister-of, Ela)

To find a daughter of Teresa, the knowledge of a predicate is needed (*is-mother-of* or reflection of this relation *is-daughter-of*). It is impossible to find a predicate, possessing only information that this predicate is a relation between two people and even knowing that the mother's name is Teresa. If the search for a predicate is processed, there are two results: *is-mother-of* and *is-sister-of* and the reasoning mechanism cannot make a valid decision which predicate to choose, therefore predicates need to be known to the reasoning mechanism in advance. In this example the list of predicates cannot be extended without informing the reasoning mechanism, adding a triple *(Person, is-grandmother-of, Person)* to a knowledgebase is simply not enough. This poses a big problem when updating knowledge in a device. The main idea is that the knowledgebase would change over time, but the reasoning mechanism stays in the same form, only interpreting all information from the knowledgebase according to fixed rules. If the reasoning mechanism is fixed, the set of predicates needs to be constant.

For simple devices the reasoning mechanism is planned to be implemented in hardware, to minimise costs and engine size, this implementation is therefore fixed. Changes can only be made in the knowledgebase file and the reasoning component need to process the changed knowledgebase if it follows the same ontology model and data formulation scheme.

7.3.5 Simple predicates

The number of predicates is reduced to three simple predicates: *is-a* to represent hierarchy of classes, *has-a* to represent an other relation between classes and *is* to associate values to instances of classes. The example from Section 7.3.4 can be translated to the simple predicates:

(Mother, has-a, Daughter)

(Sister, has-a, Sister)

(Mother, is-a, Person)

(Sister, is-a, Person)

(Daughter, is-a, Person)

(Teresa, is-a, Mother)

(Teresa, is-a, Sister)

(Anna, is-a, Daughter)

(Ela, is-a, Sister)

(Teresa, has-a, Anna)

(Teresa, has-a, Ela)

In this case when looking for a daughter of Teresa, the predicate is known (*has-a*) and the class *Daughter* is known, therefore the result of this search is Anna. The reasoning mechanism need to find triples that associate Teresa with any person from the *Daughter* class. The predicate *has-a* only points to the type of the relation.

A similar translation can be done using complex predicates:

(Mother, is-mother-of, Daughter)

(Sister, is-sister-of, Sister)

(Mother, is-a, Person)

(Sister, is-a, Person)

(Daughter, is-a, Person)

(Teresa, is-a, Mother)

(Teresa, is-a, Sister)

(Anna, is-a, Daughter)

(Ela, is-a, Sister)

(Teresa, is-mother-of, Anna)

(Teresa, is-sister-of, Ela)

This representation only adds complexity to knowledgebase, by repeating information in predicates and classes. As presenting the same information can be done with less predicates, the reasoning mechanism can be simpler. This is the main reason for choosing simple predicates for implementing the reasoning mechanism. In the case of simple predicates the reasoning mechanism does not need to be changed with new relations being added to the knowledgebase, therefore it can be implemented in hardware.

7.4 Knowledgebase division

The knowledgebase is divided into three parts: long-term knowledgebase (LTKB), short-term knowledgebase (STKB) and updating knowledgebase (UKB). The LTKB contains all the information about the device and its behaviour. If we associate knowledgebase division with description of knowledgebase content from Section 7.2, the content of differen knowledgabses is presented in Figure 23.

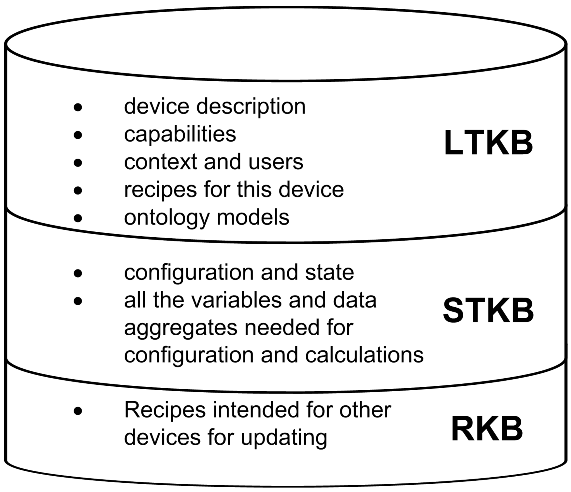


Figure 23: knowledgebase division.

The STKB in the presented architecture is equivalent of Working Memory from the Artificial Intelligence domain (see Section 4.2.2). The data stored in STKB is temporary and used to store configuration data associated with virtual devices in which the device participates, current state information that is used to adjust a device's behaviour. An example of a configuration data format is presented in Figure 24. Mandatory fields in a configuration are:

- *confName* - configuration name used to associate data,
- *virtualDeviceId* - configuration is associated only with one virtual device,

- *step* - indicates step in currently processed recipe,

- *service* - describes service offered by the virtual device.

Most of fields in configuration are flexible and any data that is somehow associated with configuration can be saved. Examples of this data are as follows:

- *person* - a particular person that is associated with the virtual device,

- *oryginalMessageId* - a message indemnification that needs to be saved under a particular configuration,

- *messageAggregate* - a structure that holds received messages.

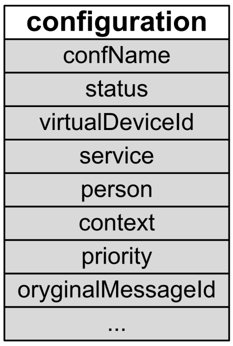


Figure 24: Configuration data.

The STKB also contains all the variables and data aggregates needed for configuration and calculations. The data in STKB can be local, intended for one configuration or global and accessible by any mechanism in the device. The UKB contains all recipes that are not intended for the device. This is a database of recipes that should be sent to other devices to enable cooperation.

The division of the knowledgebase was done to separate different types of data. The LTKB contains the main data of the device, that are not changed very often, in the Initial state of the system the STKB is empty and data is added when the device starts cooperating with others and deleted when configurations are finished, therefore STKB contains temporary data. The UKB contains recipes that are not meant to be used by the device, so separating them is the easiest way to avoid misinterpretation.

The presented knowledgebase is using an ontology for data structure and to enable semantic interoperability. An ontology offers flexible and very powerful framework for knowledgebase representation that can be easily updated and adjusted to ever-changing domains. The knowledgebase consists of ontology models, detailed device description and device behaviours customized to a particular situation. The knowledgebase presented in this chapter needs to be interpreted with the use of a reasoning mechanism called Knowledgebase Interpretation Engine (KBIE). Because the knowledge is well structured, in a simple machine readable form of RDF triples and contains all the information to make a decision, the KBIE is a simple and generic engine only following guidance from a knowledgebase. The KBIE design, development and implementation is presented in detail in the next chapter.

8 Knowledgebase interpretation engine

With technology miniaturization development, soon everyday objects will have sensing, processing and wireless networking capabilities. This will bring many objects into the computing world and enable visions like Internet of Things [66] and Smart Dust [31] to be implemented. It is inevitable that devices in a pervasive system have to process data : to make decisions, establish and manage connections, cooperate with other devices and adapt to the environment or an existing situation. The engine that enables computation for very small devices needs to be energy-efficient, cheap to produce, easy to install and make the device independent enough to work without much maintenance, following the *install and forget* approach. In this approach, after the installation process, devices need little or no maintenance in order to properly function in the system. The sensors, actuators and everyday objects that can be part of a pervasive system, might not have sufficient computing power to process additional functions, the everyday objects like light bulb might not have any processing capabilities, therefore the engine enabling cooperation needs to be self-contained, autonomous and easily attachable to a device, to control its functions. This chapter presents a context-aware generic Knowledgebase Interpretation Engine targeted for small, energy-frugal devices to enable semantic interoperability and cooperation in a pervasive environment. The Knowledgebase Interpretation Engine (KBIE) is used to reason about information represented in a knowledgebase.

This chapter is divided into seven sections. Section 8.1 describes how the engine can be used with contemporary and future devices. Section 8.2 describes the engine functionality and Section 8.3 focuses on different parts of the KBIE. Section 8.4 describes algorithms used to process a recipe and describes all capabilities of the engine used in recipes. Section 8.5 is focusing on hardware implementation of the engine and Section 8.6 describes how the KBIE was implemented with use of a CSP model. Finally Section 8.7 evaluates engines suited for complex and simple predicates.

8.1 Separation of concerns

The KBIE is designed to process entries from a knowledgebase following an ontology model. The knowledgebase is the centre of intelligence of the system, the KBIE only interprets the knowledge and navigates though the ontology tree.

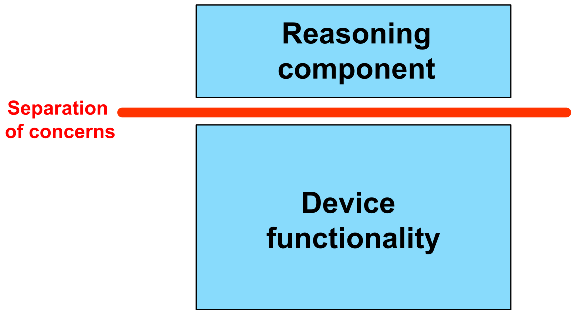


Figure 25: Separation of concerns when considering device manufacturing.

From the point of view of manufacturers the separation of concerns is very important when producing a device (Figure 25) for a pervasive system. The manufacturers of sensors and actuators are usually not computer experts and their expertise lies in the products they are making. Therefore the designed engine is an addition to existing or future devices, connected with use of a device interface. This way manufactures can produce devices and products, only providing a device interface with port specification and device description. The KBIE can use this information to influence device functionality.

8.2 Engine Functionality

The most important function of the KBIE is dealing with external signal such as messages. The action that a message triggers can be sending a message, influencing device functionality or other actions internal to the KBIE (Figure 26).

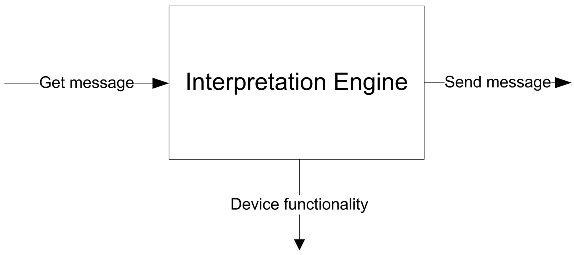


Figure 26: KBIE inputs and outputs.

Figure 27 illustrates the KBIE state diagram. When the message first arrives it has to be parsed. The messages can vary in size and field types. The KBIE is designed to interpret any message type, as long as it is informed about its structure before it gets the message and the first field indicates the message type. All message structures the device needs to know are represented in the knowledgebase. Modifying or extending knowledgebase information about the message format enables the engine to parse any type of message.

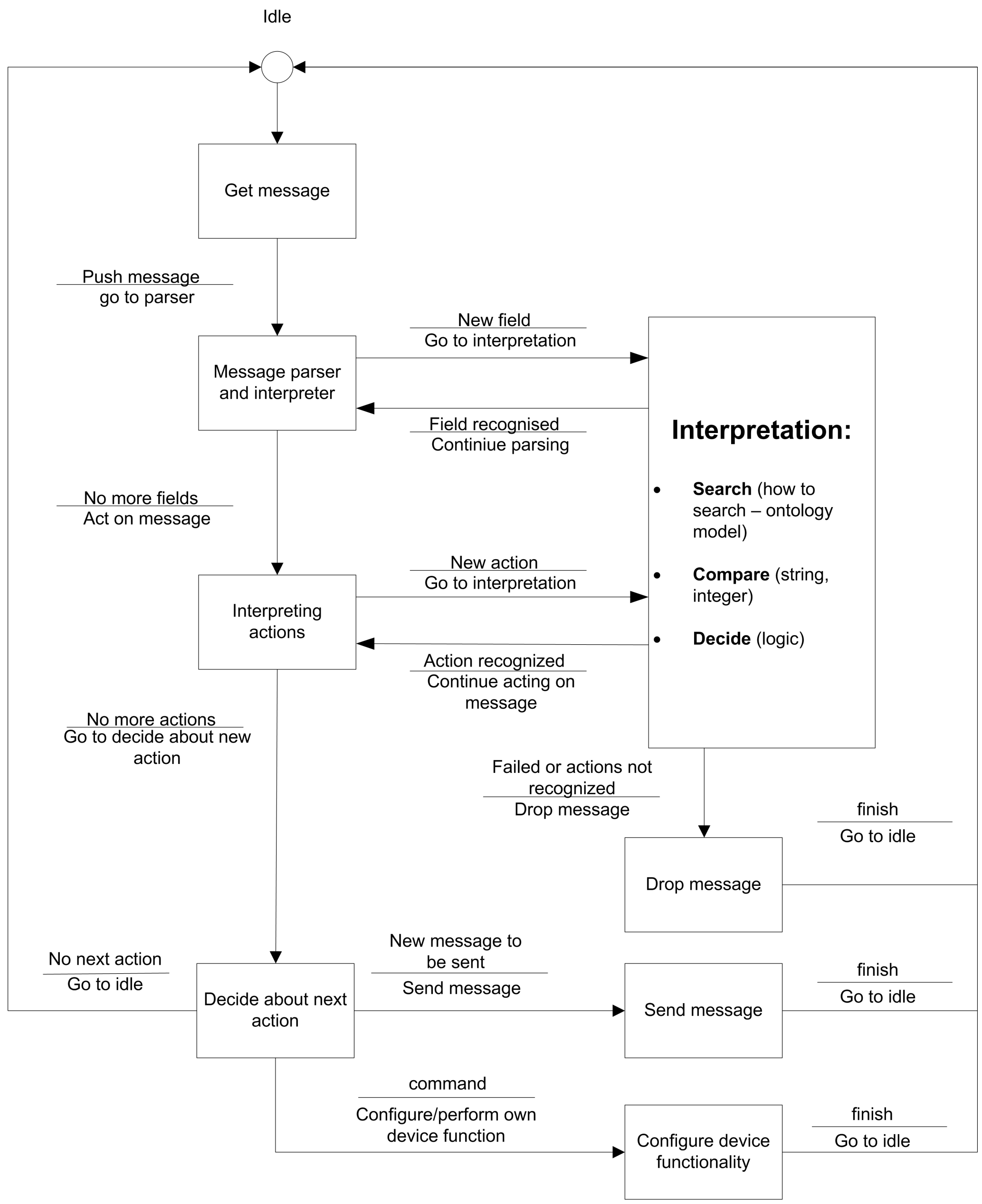


Figure 27: Detailed state diagram.

The first field in the message is a *messageType*, the first step of parsing interprets only message type to match it with message types that are represented in LTKB. If a match is found the message is parsed, so written in a format understandable by the engine and easily accessible within one configuration. Once the message is parsed it needs to be interpreted. The message interpreting mechanism depends on message type. Once a message is interpreted actions associated with a message type and content needs to be executed. Most messages either run a new recipe or continue the recipe associated with a particular configuration that includes for example sending a new message, configuring device functionality or other actions associated with a recipe. If the message is not parsed or interpreted properly, or is not designed for the device, the KBIE drops the message.

8.3 Device Architecture

The overall device architecture consist of a knowledgebase, KBIE, device interface, communication interface and device functionality as presented in Figure 28. The device functionality is built by the device manufacturer, a device interface must be specified in order to influence the function of a device, for example how to turn off a light bulb. The communication interface is receiving and sending bits of information trough any medium, the communication mechanism is required to be broadcast based. The communication interface can be part of the KBIE or the device functionality. The KBIE is designed to receive and send signals, interpret knowledge from knowledgebase and influence device's functionality. The knowledgebase was described in Chapter 7, in this chapter the KBIE is presented and its functions for navigating and interpreting the knowledgebase.

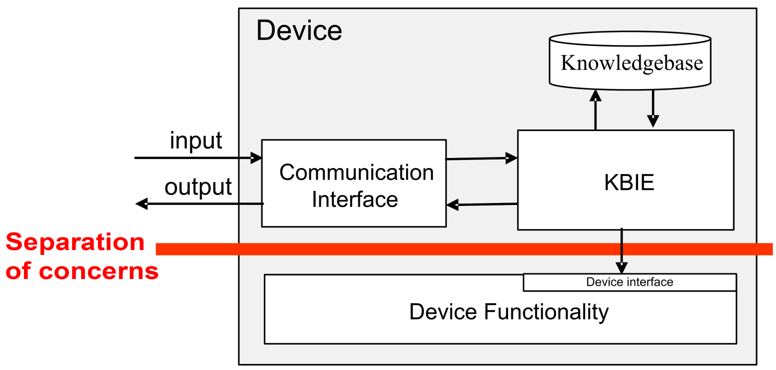


Figure 28: Smart device architecture.

The detailed device architecture for a pervasive system is presented on Figure 29. The KBIE is divided into three parts: Control Unit, Poke and Knowledgebase

Fetch.

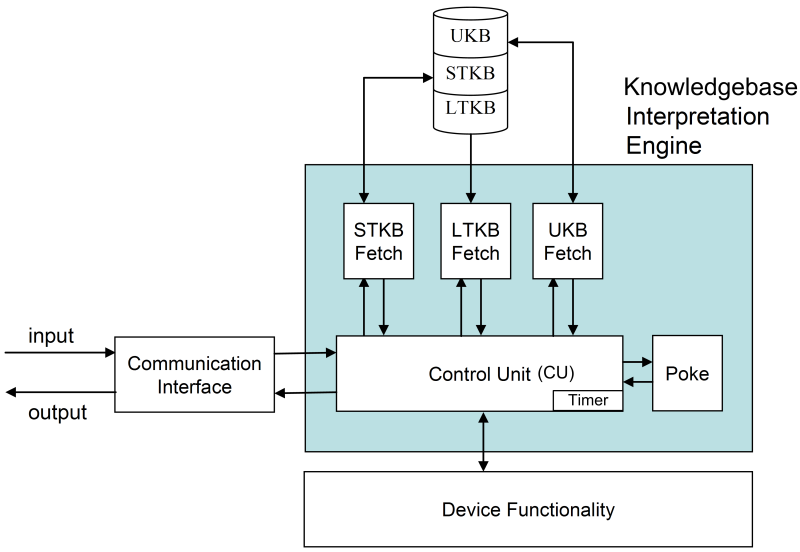


Figure 29: Device with KBIE.

8.3.1 Control Unit

The Control Unit is the main part of the KBIE, it receives all inputs from Communication Interface, Device Functionality and internal timers associated with scheduled tasks. The Control Unit reacts on input and first interprets the request with use of entities in the knowledgebase and makes decisions to act upon a request.

The Control Unit (CU) is always ready for input signals from five KBIE entities: device input, timer for single scheduled tasks, timer for constant scheduled tasks, device functionality and Poke mechanism. On device input, the CU is ready to receive messages. The CU first checks if the signal is a message, if so message is parsed. The message types and fields are represented in the knowledgebase, therefore the parsing mechanism is automatic. The message is saved in STKB to be used later on. The description of a search message from the knowledgebase is presented in Table 1.

When the message is parsed, the CU continues to interpret the message. Depending on the message type the CU performs different actions. For a Search message (see Section 5.4.1) the CU first checks if the device provides requested service and understands the context for any person specified in the message fields. If so the configuration data is saved to STKB and an appropriate recipe is searched and executed. When the recipe finishes, the configuration is set to status *engaged* that

(searchMessage,is-a,Message)	Defining message type.
(messageField,is-a,smartDataField)	
(HeaderF0,is-a,messageField)	Defining message field.
(searchMessage,has-a,HeaderF0)	Associating message and field.
(HeaderF0,has-a,idF0)	Defining field ID.
(idF0,is-a,id)	
(idF0,is,0)	
(HeaderF0,has-a,nameF0)	Defining field Name.
(nameF0,is-a,name)	
(nameF0,is,MessageType)	
(HeaderF1,is-a,messageField)	Defining another message field.
(searchMessage,has-a,HeaderF1)	Associating another message field with message.
...	

Table 1: Search message representation in knowledgebase.

indicates that the virtual device has been established.

To answer the search message, an acknowledge message is issued (see Section 5.4.3). When the CU receives the acknowledge message, first it checks if the message is for a virtual device in which this device participates. In order to do this, the CU searches for a configuration in STKB that is associated with the requested virtual device. If the configuration is found, the CU checks if this configuration is awaiting for a search message, it therefore searches for the Original Message Id data in configuration and checks it against the field from the acknowledgement message. If all checks are successful, the message is saved in a message aggregate in the appropriate configuration.

According to the protocol presented in Section 5.5 next an acceptance message is issued. The CU from a device that is receiving the acceptance message (see Section 5.4.2) first checks if the message corresponds to any configuration saved in STKB, therefore checking the virtual device ID. If the information about a particular configuration is found, the message is saved in an aggregate associated with this configuration.

The CU can also receive a control message (see Section 5.4.6). As the control message is only associated with one virtual device, the CU checks if this device participates in the given virtual device and if the configuration status is *engaged*, implying the virtual device has been established. When the message is accepted, the CU checks if commands from the message are understood by the device. Next the CU searches for a recipe that interprets data contained in the control message.

While participating in a virtual device, a device can receive a data message (see

Section 5.4.7). The data can be of any type, therefore the CU needs to asses if the type is recognised by the device and if there is a recipe that deals with the data.

A destroy message (see Section 5.4.5) can be issued, when a virtual device needs to be destroyed. The CU first checks if the destroy message is intended for this device, if so all configuration data is deleted from STKB.

To run a recipe internally, therefore without any input signal, an internal message is used (see Section 5.4.8). This message is forwarded to a device and treated by the CU as an input request. The CU is treating this message as a search message and searches for the appropriate recipe to run. The issuing recipe can either wait or continue without notification from the executed recipe.

The CU can also receive a shout message (see Section 5.4.4) that is a request to all the devices that posses the required service and context to report to the issuing device, in response to this message an acknowledgement message is sent. This mechanism enables what sort of devices there are in the system and the virtual devices in which they are involved.

The update and transmit mechanisms are hard-coded in the engine, in future implementation it is planned to use appropriate recipes to enable an updating mechanism and recipe exchange.

Messages are the most important signals that are received by the CU, another type of signal is a timer signal. Timers are used to control interval between events. Timers are set from values in the knowledgebase and are associated with single or multiple scheduled tasks. A single scheduled task can set a timer to send some signal at some time in the future and the alarm will ring only once. An example of single signal is waiting for acknowledgement while configuring a virtual device. A multiple scheduled task is a task that is executed constantly and its started and stopped by recipes. An example of multiple task is the functionality of a sensor, that is sending data to other devices at some predefined interval.

The CU can also receive signal from Poke, the signal is numeric and represents a configuration to be reactivated. The Poke mechanism is described in Section 8.3.3.

8.3.2 Knowledgebase Fetch

The Knowledgebase Fetch blocks are responsible for managing different parts of the knowledgebase, namely long-term, short-term and update knowledgebase (Section 7.4). All Knowledgebase Fetch mechanisms can query different knowledgebase file. The query method is similar to RQL (RDF Query Language [32]) and SPARQL [50] and uses simple matching to find the requested RDF triples. The basic Knowledgebase Fetch requests: add, remove and search are presented in Table 2.

Operation	Function	Translation to SQL statement
Add a triple to LTKB	kbQuery(0,rdf('Anna','is-a','Person'));	INSERT INTO ltkb (subject, predicate, object) VALUES ('Anna','is-a','Person')
Remove a single triple from LTKB	kbQuery(1,rdf('Tom','is-a','Dog'));	DELETE FROM ltkb WHERE subject='Tom', predicate='is-a', object='Dog'
Remove many triples from LTKB	kbQuery(1,rdf('Tom','has-a',''));	DELETE FROM ltkb WHERE subject='Tom', predicate='has-a',
Search for a tripple in LTKB	kbQuery(2,rdf('Anna','is-a','Person'));	SELECT subject, predicate, object FROM ltkb WHERE subject='Anna', predicate='is-a', object='Person'
Search for a sublect in LTKB	kbQuery(2,rdf('Anna','',''));	SELECT subject, predicate, object FROM ltkb WHERE subject='Anna'

Table 2: Examples of basic knowledgebase operations.

With the use of simple queries its possible to execute more complicated requests on the knowledgebase. If a search triple needs to have subject and object of some type, two types of queries are used: *MatchSubjectInModel(modelRdf,subject)*, when the subject of a triple is known or *MatchObjectInModel(modelRdf,object)*, when the object of a triple is known. Consider a knowledgebase representing information about family members consisting of triples:

(Mother, has-a, Daughter)

(Mother, is-a, Person)

(Daughter, is-a, Person)

(Teresa, is-a, Mother)

(Anna, is-a, Daughter)

(Margaret, is-a, Daughter)

(Teresa, has-a, Anna)

(Teresa, has-a, Margaret)

When searching for all daughters of Teresa, the association needs to be expressed in triple form model *(Mother, has-a, Daughter)* and the search criteria is *subject='Teresa'*. Therefore a request for subject is expressed as follows: *MatchSubjectInModel(rdf('Mother', 'has-a', 'Daughter'),'Teresa')*. The steps in this requests are shown in Table 3. Similarly, the *MatchObjectInModel* mechanism can, for ex-

Query	Explanation
kbQuery(2,rdf('Teresa','is-a','Mother'))	Checking if the query is correct, so if Teresa is a mother.
kbQuery(2,rdf('Mother','has-a','Daughter'))	Checking if the model triple is correct.
kbQuery(2,rdf('Teresa','has-a',"))	Searching for for all Teresa's associations, this operation returns *Anna* and *Margaret*.
kbQuery(2,rdf('Anna','is-a','Daughter'))	Checks if returned entity is of type *Daughter*.
kbQuery(2,rdf('Margaret','is-a','Daughter'))	Checks if returned entity is of type *Daughter*.

Table 3: Steps of MatchSubjectInModel mechanism.

ample, find Anna's mother. This mechanism allows narrowing the search to some model entry. Thus the required operations are very simple and easily implemented.

8.3.3 Poke

The Poke block is used to reactivate recipes that have been suspended due to running new recipes. The Wait-Poke mechanism is shown in Figure 30. The first case occurs when results from *Recipe 2* are important to be able to finish *Recipe 1*, for example if the virtual device established by *Recipe 2* is necessary to establish a virtual device with use of *Recipe 1*.

In Case 1 *Recipe 1* needs to wait wait for a signal (called *poke*) from *Recipe 2*. *Recipe 1* cannot proceed without the signal. In Case 2 *Recipe 1* runs *Recipe 2*, but the function hat is required is optional and not important for *Recipe 1*, therefore *Recipe 1* can be finished even if *Recipe 2* has been dropped. The *poke* signal is

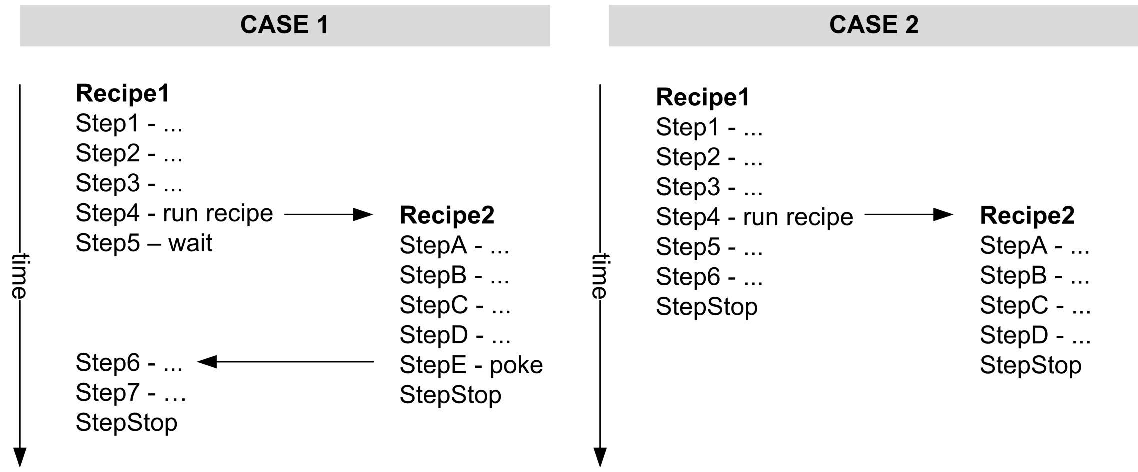


Figure 30: Two cases of running recipe within a recipe: Case 1 with use of Wait-Poke mechanism; Case 2 without waiting.

numeric and contains a virtual device identification number of the configuration that has been finished. The activated configuration (associated with *Recipe 1*) has this number saved, therefore it is the only configuration that is reactivated.

8.4 Processing recipes in KBIE

A recipe is processed step by step. A function to process one step in the KBIE is called a *processStep*. The function input is the configuration name and output a Booleand value that indicates if the step was successfully finished. If the value is *true* the engine keeps on processing the recipe, if *false* the recipe is suspended. The flowchart of the *processStep* implementation is presented in Figure 31.

Every step is able to run one capability, capabilities determine what primary action the step is performing. The details of actions are specified in the step. For example a step that is sending an acknowledgement message is presented in Table 4 (for the whole lamp recipe see Appendix A).

The considered step is called *Recipe1S1* (Table 4 line 31) and it is associated with the action called *RespondConfigRequest* (line 32). The action is an aggregate for capability and specific data, in case of action *RespondConfigRequest*, it is information about a message that should be send *message1* (line 35) in this step. The *message1* is an acknowledgement message (line 36). The action posses information about one of the message (line 38) with *id* equals '5' (line 41) and *name* equals 'ackResponse'. The action can have information about any number of fields in the message, all

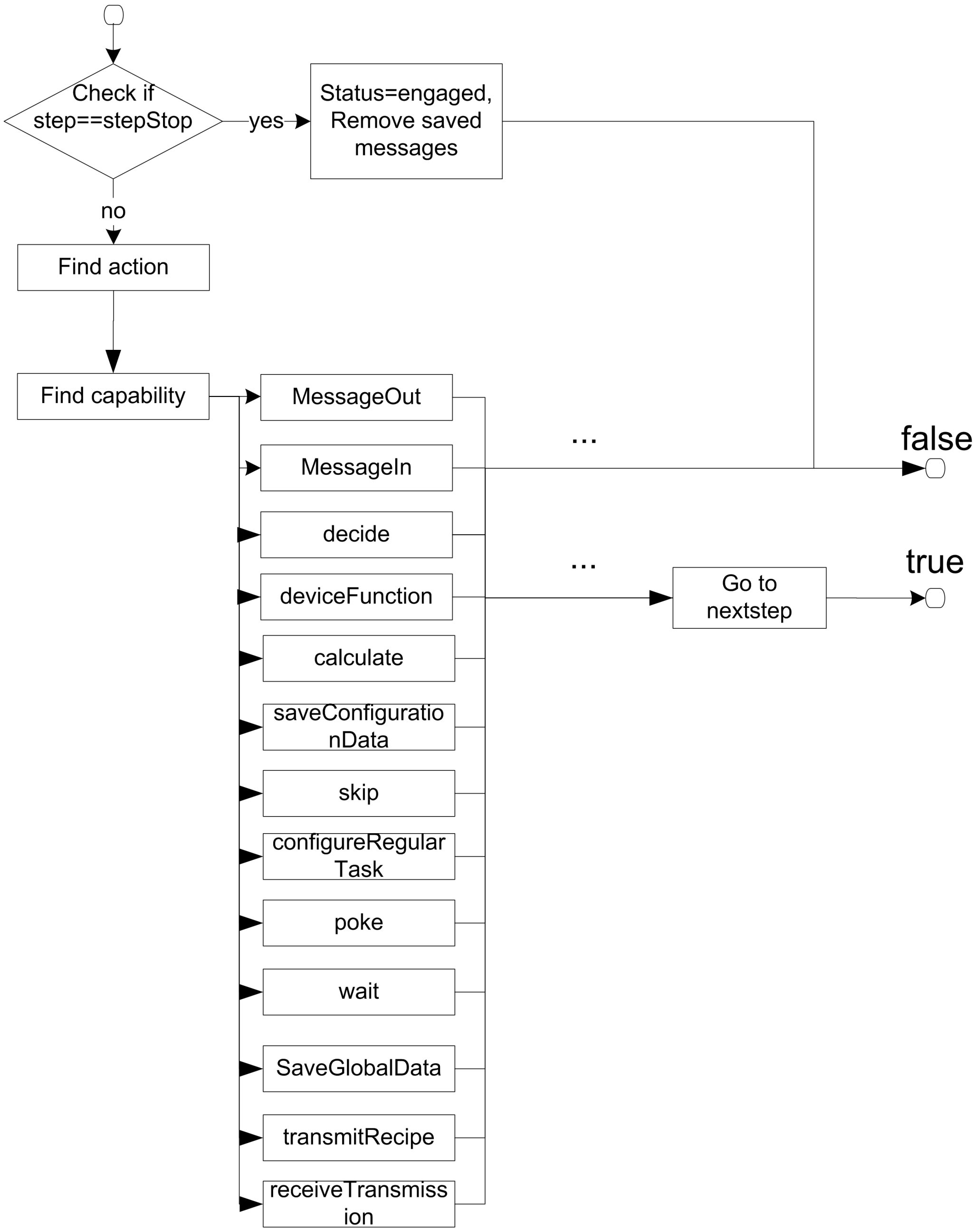


Figure 31: Flowchart representing a *processStep* function.

other fields are filled by the message building mechanism and are taken from the configuration and global data.

In the final version of the engine thirteen capabilities were implemented. This enables all devices running on the KBIE to: receive and send messages (messageOut, messageIn); save local for configuration and global for device data (saveConfigurationData, saveGlobalData); analyse messages using rules (decide);send commands to a device (deviceFunction); perform calculations and comparison (calculate); trigger running another recipe and reactivate recipes (poke, wait); skip steps with a boolean condition as an implementation of *if-then-else* statement(skip); simple streaming

31 . (Recipe1S1 , is-a , step)
32 . (Recipe1S1 , has-a , RespondConfigRequest)
33 . (RespondConfigRequest , is-a , action)
34 . (RespondConfigRequest , has-a , massageOut)
35 . (RespondConfigRequest , has-a , message1)
36 . (message1 , is-a , ackMessage)
37 . (message1 , has-a , field1)
38 . (field1 , is-a , messageField)
39 . (field1 , has-a , id2)
40 . (id2 , is-a , id)
41 . (id2 , is , 5)
42 . (field1 , has-a , value2)
43 . (value2 , is-a , value)
44 . (value2 , is , 0)
45 . (field1 , has-a , name2)
46 . (name2 , is-a , name)
47 . (name2 , is , ackResponse)

Table 4: Step *Recipe1S1* from *Recipe1* in lamp's knowledgebase (Appendix A).

recipes triple-by-triple between devices (transmitRecipe, receiveTransmission); configure a regular task to trigger at a certain rate (configureRegularTask).

8.4.1 messageOut

The capability *messageOut* is responsible for sending a message. The message that is being sent first needs to be built. To build a message first the message type is needed, therefore the step needs to provide this information. The step can also suggest field values, other values are taken from configuration data and global data for the device. When the message is successfully constructed and sent, the engine saves data needed for processing the next step and the block is finished with a *true* value.

8.4.2 messageIn

The second capability managing messages is *massageIn*. This capability is responsible for receiving messages. According to the architecture presented in Section 5 a device needs to wait some time to receive a message within the configuration mechanism. The time to wait depends on a recipe and it has an assigned value in the step. The *massageIn* sets the timer to retrieve a value only once as there can be only one timer for a virtual device. After the initial phase of setting the timer, the configuration needs to say in the step associated with capability *massageIn* because the device waits for messages to be delivered. Only the timer can finish this phase of receiving a message. When a new message arrives, the configuration is still in a step associated with the *massageIn* capability, as the timer has already been set, the device saves the incoming message within a message aggregate in this configuration. This block always returns a *false* value, that is stopping the engine from processing another step. When the set timer triggers, the engine starts processing the next step in the recipe.

8.4.3 decide

Decisions in the KBIE are associated with messages received by the device. There are two types of decisions, based on a rule or acceptance. If the *decide* capability has a rule associated with it, the step analyses only the field specified by the rule. Only when the field is accepted, the message is remembered within a message aggregate associated with a current configuration. After checking all messages, the KBIE checks if there are any messages saved, if so the engine continues to the next step and returns true. The second type of *decide* capability only check if there are any messages received by the device associated with this configuration. If there is one or more messages the block initiates the nest step and reruns true.

8.4.4 deviceFunction

To influence device functionality a step needs to be associated with the *deviceFunction* capability. This block is looking for commands and values to send to the Device Functionality component (Figure 29). First the engine checks if the commands are understood by the engine, therefore looking for triples *(”,’is-a’,’command’)* (for an example see Appendix A, listing 15-18). When a command is found, the block searches for a specyfic command that is understood by the device (for an example see Appendix A, listing 20-24) and sends a pair *(command,value)* to the Device Functionality.

8.4.5 calculate

The *calculate* capability is responsible for comparing, calculating or finding a value in a pre-defined function. The engine only supports two attribute operations. The *calculate* capability uses a structure called *calculationData* to represent information that needs to be calculated.

calculationData
calculationType
attributeType
attributeX
attributeY
operator
outcome

Figure 32: Calculation Structure.

The *calculationData* structure is presented in Figure 32, where fields are as follows:

- *calculationType* field is one of values *compare*, *arithmetic* or *function*,
- *attributeType* is either *Integer* or *String*,
- *attributeX* and *attributeY* are attributes of the function or calculation,
- *operator* indicates the operator or function used in this calculation,
- *outcome* is a name of a variable to save the calculation result.

An example of use the *calculate* capability is a step *Recipe3S0* in *Recipe1* (see Appendix A listing 261-320).

8.4.6 saveConfigurationData and saveGlobalData

The *saveConfigurationData* capability saves a value with the name in the STKB under the current configuration. The capability *saveGlobalData* is very similar and saves a variable and specified value in STKB as global data. For example to save data of type NewData with value of '5' under a configuration name 'config40' following triples needs to be added to STKB:

configuration	has-a	NewData
data1	is-a	NewData
config40	has-a	data1
data1	is	5

The first triple (configuration, has-a, NewData) is extending the ontology model allowing associate configuration with new data type. Next triples associate new variable 'data1' with the requested data type, configuration and value.

8.4.7 skip

The skip mechanism is presented in Section 7.3.2, the capability *skip* directs the engine to the required step in the recipe.

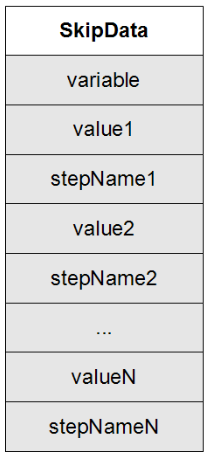


Figure 33: Skip Structure.

Figure 33 presents the data that is associated with the capability. The engine needs a variable to decide which step it should go to, values: *value1, value2, ..., valueN* direct the engine to steps: *stepName1,stepName2, ..., stepNameN*. In the example presented in Appendix A (listing 261-320) if the variable *teporaryData2* equals 'TRUE' next step is *Recipe3S5*, if it is equal to 'FALSE' the engine proceeds to step *Recipe3S13*.

8.4.8 configureRegularTask

A step using capability *configureRegularTask* is responsible for setting a timer for a regular task. When the timer triggers all recipes that are responsible for a regular task are run. An example of use of this step is setting a sensor to send a value of a reading every 2 seconds.

8.4.9 poke

The *Poke* mechanism was described in Section 8.3.3. The step using capability *poke* sends a signal to Poke block to reactivate a recipe that is waiting for an input form the current virual device.

8.4.10 wait

The *wait* capability simply stops the engine to process another step. This behaviour is part of Poke behaviour described in Section 8.3.3.

8.4.11 transmitRecipe

A step using capability *transmitRecipe* is designed to find an appropriate recipe and send triple-by-triple to all devices that requested an update. This capability is forming and sending transmit messages (see Section 5.4.10) from triples found in the UKB.

8.4.12 receiveTransmission

The capability *receiveTransmission* enables a device to receive triples for update, this capability changes the configuration state to *transmission*. When a transmit message arrives the data is next saved in LTKB. The end of transmission is detected when $TransmissionSize = TripleNumber + 1$.

The described set of capabilities is sufficient to perform all planned scenarios only by changing recipes that are run on devices, without modification of KBIE.

The engine relies on the knowledgebase providing data, therefore if a triple that is expected is not found, the engine drops processing a step. Every box represented in Figure 31 actually have conditions itself as presented in Figure 34. The engine does not proceed when there are mistakes in the knowledgebase, as the knowledgebase is the only source of information for a device. If a recipe is corrupted, it is not executed.

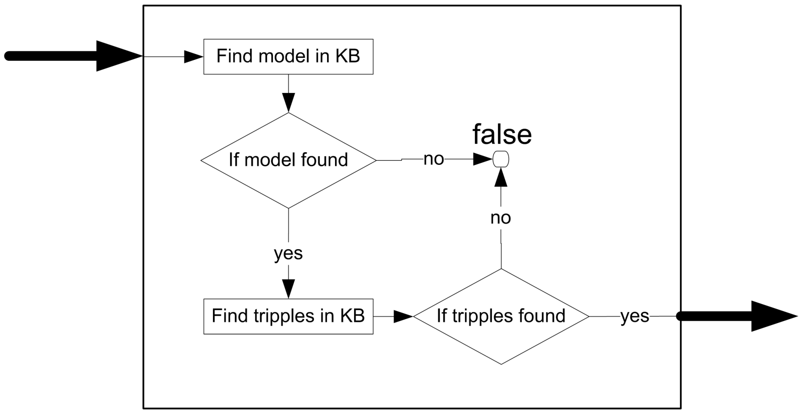


Figure 34: Process step conditions when triple not found.

8.5 Targeting hardware

A device that is a part of the developed system is considered to be very small and have limited computation power. One of the project objectives is to determine if it is possible to create a simple engine to interpret and maintain information from a knowledgebase without using a general purpose processor. The knowledgebase interpretation engine is intended to be developed in hardware, so the number of functions that the engine will perform has to be determined. Research questions referred to this issue are: *Is it possible to produce a way of combining operands and functions to a small number? For this type of system can we identify the required functions and reduce them to a small number?*

If the number of functions in the engine would be equal to functions performed by a processor, there is no need to design an engine that will support knowledgebase interpretation. If the number of functions can be reduced, a targeted engine can be built at low cost and can be attached to a device to form a pervasive system. The idea is to put all intelligence in a knowledgebase, so its content informs the device how to behave in different situations. The engine will only be used to interpret this knowledge and help a device act upon instructions contained in the knowledgebase. Therefore the engine would only perform limited functions to enable device collaboration, devices existing in a pervasive system can have their own engines/processors to perform tasks specific for them. The basic functions and their use in the engine are presented in Table 5.

String operations are included in the engine because of the content of the knowl-

Basic Function	Example in KBIE
String compare (==)	Comparing entities in knowledgebase,
String concatenation (|)	Used when creating unique names for entities in knowledgebase,
Integer compare ($==, <, \leq$)	Analysing data input and messages,
Integer arithmetic	Analysing data input.

Table 5: Basic functions identyfied in KBIE.

edgebase. A translation can be applied to convert String content of the knowledgebase to Integer or any other numeric type. After knowledgebase translation, the engine only needs to deal with two sets of instructions: Integer compare and Integer arithmetic. Some complicated calculations can be presented as tables of values in the knowledgebase. For example functions to regulate emergent behaviour of a lamp in Scenario 6.2.2 are written using selected points (x, y) in the knowledgebase presented in Appendix A (lines 1182-1895). The presented engine performs a subset of functions of a general purpose processor and additionally the integer arithmetic can be substituted with use of functions described in knowledgebase.

8.6 Engine implementation

The engine was designed and implemented using CSP as an underlying model. The use of CSP models for this architecture is justified by its parallel execution capabilities that can reflect the parallelism of hardware. The CSP model was previously successfully used to verify protocols [53], scenarios from the case studies are simulated with use of CSP implementation for Java.

The idea is to use concurrent programming techniques to implement the infrastructure of a lighting system consisting of many devices with different capabilities. We can consider the system from two points of view. First is the view of a system as a whole, therefore all devices that are present in a smart space. Second view is of a device itself, therefore the engine that can process requests sent to a device and influences the devices functionality, and knowledgebase management.

From the first view we can observe how devices react to different environment conditions and user needs and also how devices are grouped for cooperation. This view is presented by a simulation developed in this project (Figure 35). Devices in the system are represented in Figure 35 as a set of processes labelled as $D_1, D_2, ..., D_n$, B represents a broadcast mechanism explained in Figure 36, a *SGI* component (Simulation Graphical Interface) is a process responsible for gathering and displaying data generated by the system.

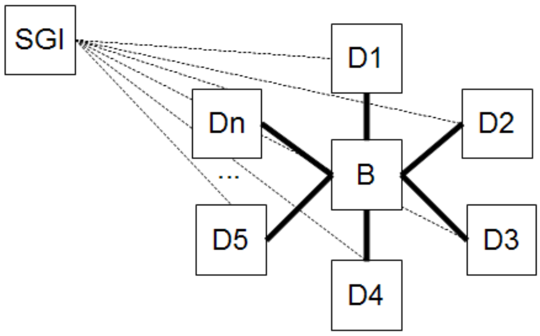


Figure 35: Simulation top level.

The smart lighting system chosen to illustrate a context-aware distributed system consists of light switch, different kinds of light sources and sensors (Figure 14, see Chapter 6). The system as a whole is implemented using a simulation, but work on transferring the implementation to hardware (microcontroller board for every device) is in progress. The present simulation was implemented in JCSP (Communicating Sequential Processes for Java) [3]. This language supports concurrent programming; it is useful when simulating devices running simultaneously. Devices are presented as CSP processes and communication links are implemented using CSP channels. Use of JCSP enables simulating a network of many devices running in parallel and communicating by sending messages.

The simulated system is using broadcast to propagate messages. A broadcast mechanism do not exist in CSP, the Broadcast component is added to transmit messages in the system. Channels between devices and Broadcast component need to be buffered, so the locking mechanism does not deadlock the whole system, for this purpose *one2one* buffered channels were used from JCSP library *jcsp-1.1-rc4* [68]. As shown in Figure 36, the message output from the Device 1 is sent to Broadcast component. The Broadcast component is next propagating the message to all the devices, including the sending device.

The second level of the project is the device itself and all mechanisms in this device (Figure 37). A device is designed to understand protocols to enable collaboration in the network but also to interpret the knowledge structures enabling it to make intelligent decisions. As the designed engine is generic, devices react according

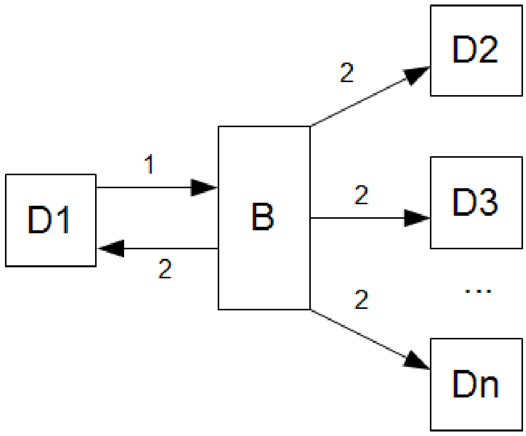


Figure 36: Broadcast mechanism implementation in CSP.

to knowledge stored in a knowledgebase, which is specific for a device. In Figure 37 CSP diagrams of CU, Knowledgebase Fetch (LTKB,STKB and RKB)and Poke are presented.

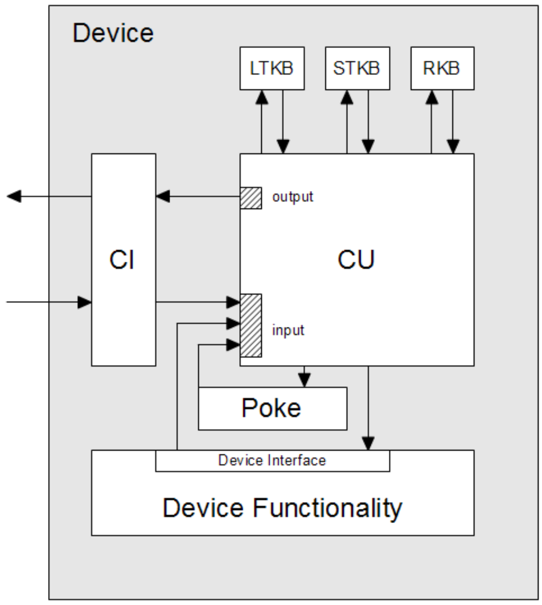


Figure 37: CSP implementation of the KBIE.

For the purpose of the simulation, the Communication Interface and the Device Functionality were also implemented as shown in Figure 37. The Communication Interface receives and sends messages in the system. The Device Functionality is simply designed to receive a pair $(command, value)$ from the CU and, on a request, send information about changing behaviour to the Simulation Graphical Interface.

8.7 KBIE with complex and simple predicates

The change of knowledgebase format from complex to simple predicates has an influence on the KBIE. The CU deals with the actual content of triples, therefore only this part of the KBIE changes when translating the knowledgebase. As the simple predicates are less descriptive, the size of the knowledgebase decreases when translating from complicated predicates, as more triples are needed to describe a relationship between entities.

Comparison criteria	Complex Predicates	Simple Predicates
Size of KB in triples after one-to-one translation	276	322
Size of KB in triples after moving message description and functions to KB		1830
Size of the KBIE in SLOC	∼ 2800	∼ 1900
Knowledgebase accesses for one recipe	∼ 900	∼ 21000

Table 6: Two styles of knowledgebase construction comparison.

As presented in Table 6 the translation from complex to simple predicates and moving some functionalities and calculations to the knowledgebase decreases the size of the knowledgebase while increasing the Source Lines Of Code (SLOC) in the CU from the engine.

The fact of moving message format and some notations for calculations to the engine makes the KBIE design more flexible by use of any message format and combination of simple calculations. This translation influences the knowledgebase accesses, because the engine has to ask the knowledgebase for much more information than it was before improvement. The Figure 38 shows the output of jconsole (Java Monitoring and Management Console) for the engine processing complex predicates and fixed messages and functions. The CPU usage is mainly stable around 10-20% , cahnging to around 60% when a recipe is executed. In Figure 39, presenting system with improvements, the CPU usage is almost constant around 80%. This shows that the knowledgebase access increased noticeably and the PC has problems with running all 22 devices with KBIE supporting RDF triples simple predicates.

The increase in knowledgebase accesses noticeably slows down the simulation. The target implementation in hardware, the knowledgebase search speed can be increased by parallel database access. The work on optimising the speed of Knowl-

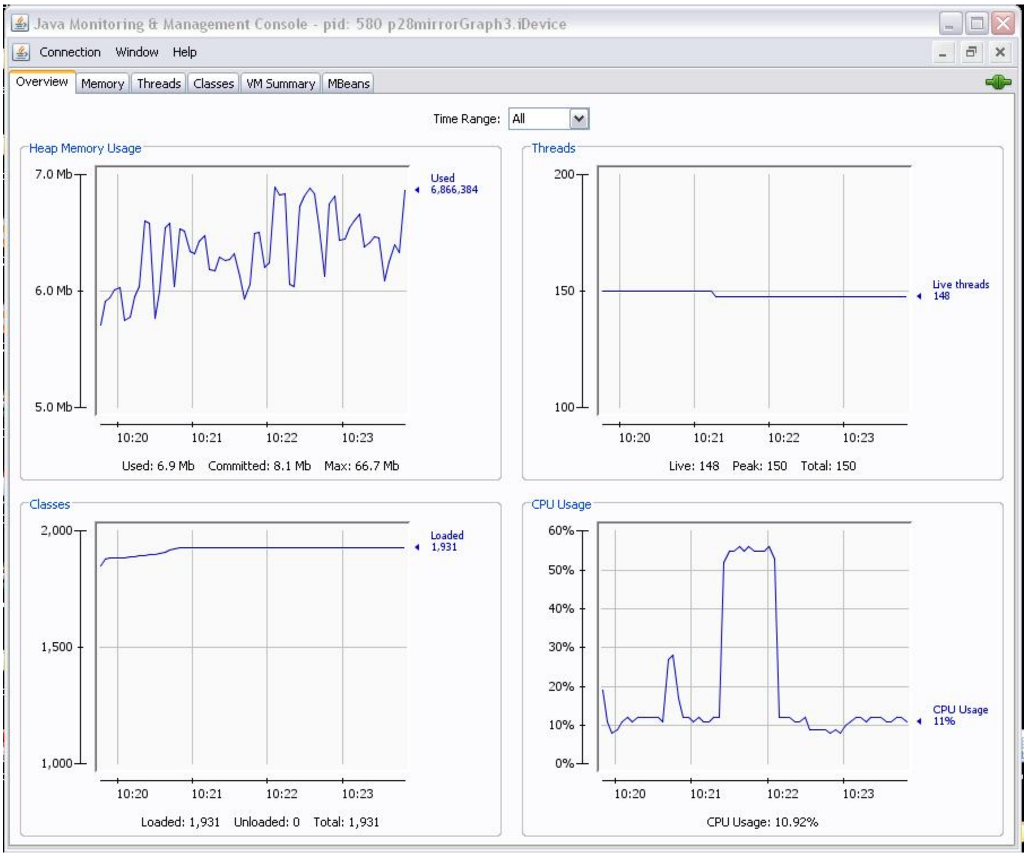


Figure 38: Test1.

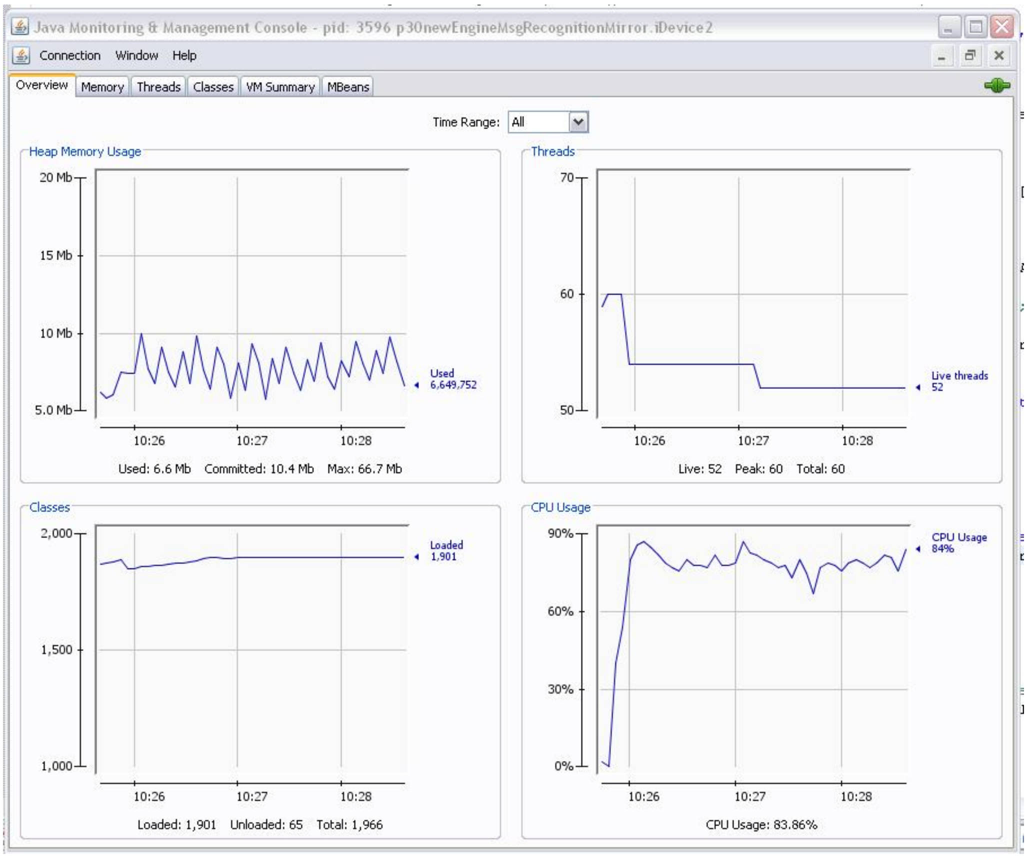


Figure 39: Test2.

edgebase Fetch component is being undertaken at Technische Universiteit Eindhoven (*www.tue.nl*), see Section 10.2.

The Knowledgebase Interpretation Engine is designed to run any recipe that is designed according to the recipe ontology and follows fixed rules, therefore it is possible to install the same KBIE in different devices only alternating the content of the knowledgebase. Examples of different recipes triggering a specific devices behaviour are presented in the following chapter.

9 Experiments

The implementation developed for the purpose of this thesis was tested in order to give information about the quality of the produced code for the future implementation of KBIE and to check if the system performs as expected. Case studies were designed to present different possibilities of the lighting systems presented in this chapter. Experiments and functional tests of the claimed behaviours are presented in this chapter.

Experiments were performed during different stages of the project. The initial experiment described in Section 9.1 took place in early stage of the project at Edinburgh Napier University and revealed some problems with the interoperability and usability of the solution for low computational power devices. Experiments with the new KBIE architecture on the example lighting system are presented in Sections 9.3, 9.4 and 9.6 and were performed at NXP Semiconductors.

9.1 Initial Experiments

The initial experiment was performed with a network of autonomous Personal Digital Assistant (PDA) devices connected in an ad-hoc manner over a TCP/IP network described in [35]. All devices are addressable, in this case it is an IPv4 address. Devices spontaneously appear and disappear from the space. Every device offers services that need to be discovered by other devices in order to co-operate. There is no central repository of services available in the space. The repository of device IP addresses is available only for the purpose of establishing connections. The dynamic connection capability in *net2* package of JCSP library [68] requires an actual IP address of the existing device on the network. Therefore all devices entering the space are required to report to the IP address repository.

The expected behaviour of the system is that after a configuration stage all devices are informed about services available on other devices and they posses an up-to-date list of available devices. This task needs to be done without using much of the network bandwidth.

When a new device enters a space it needs to register in the IP address repository server. The list from the server is sent to the new device in order to discover devices in the existing network. After the initialization phase, the device manages the list of available devices with a device discovery mechanism described in detail in [35].

The device uses the list to inform all the devices in the network about its existence and to gather information about the existing network. Therefore a new device sends a mobile agent for a trip described using a devices list to update information about the network. The agent is defined and created in a device process, sent to the other device, connected to it and run in parallel with other processes on the device. The agent connected to the device communicates with, exchanges data and informs the device its next destination. The final destination of the agent is the device that sent it. Data about devices from the network is updated in this way.

The system is behaving as expected and a single request is used to perform two-way discovery while keeping data about other devices private.

The system uses mobile processes to enable moving an object of a mobile process with all data that belongs to it to another device, connect it to the network of existing processes and run in parallel on a device. In order to use this capability a piece of code: the mobile process needs to be written in Java, the receiving device needs to connect the process in the predefined way, the process needs to be an object of a predefined class known to the device and the device needs to run a Java Virtual Machine (JVM) to be able to run Java code.

The constraints posed by this solution are significant. In the next experiment the idea of sending executable code between devices is preserved, but some constrains have been removed. The language was changed to a simple language based on an ontology and written using RDF, processes are called recipes. The recipe exchanged between devices is only constrained by the ontology model (that can be sent with the recipe) and capabilities that the engine can run. The recipe is simply appended to the knowledgebase of the device and can be run immediately. The need to run a JVM on every device was removed, as no Java is used, and replaced with a Knowledgebase Interpretation Engine targeted to be implemented in hardware, therefore suitable for small and constrained devices.. The experiment presenting knowledgebase code mobility is presented in Section 9.6 describing Experiment Three.

9.2 Experiments with the smart lighting system

Experiments with the smart lighting system were planned to realise scenarios presented in Section 6.2 and embedded in the environment as presented in Figure 14 (Chapter 6).

The primary functional testing described in this chapter is a system testing, therefore the complete system was tested in order to determine if requirements of particular scenarios are met. The secondary functional tests were limited to the KBIE and testing the engine function with respect to the complexity of: the environment (number of devices), the network (size of the virtual device) and the device function. The scalability of the developed solution was also tested by performing scenarios of different complexity using the same engine implementation. The system was also tested to trigger errors and exceptions. Tested errors were associated with receiving data and checking knowledgebase entries.

9.3 Experiment One

Experiment one is based on Scenario One (Section 6.2.1). In this scenario a switch is cooperating with lamps to provide user-defined lighting settings. The switch is able to initialise and control a virtual device of sixteen lamps. User's specific context are displayed in switch graphical user interface (GUI). The GUI used for tests is presented in Figure 40, which shows the available buttons on the GUI.

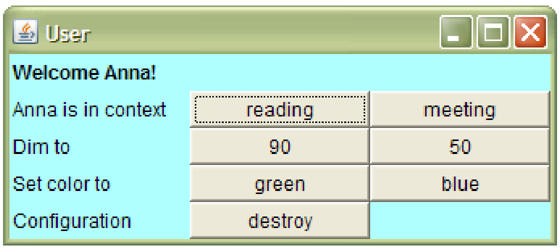


Figure 40: Graphical User Interface in Experiment One.

The switch welcomes the user and presents available contexts. In this case context is a predefined lighting setting, *Reading* sets all lights to light-yellow color and different dim level, depending on pre-defined settings. The context *Meeting* sets the for lights in the middle of the room to 100% and turns off rest of the lights. The

GUI enables the user to set color and dim level, in case he/she is not happy about the default settings. The GUI displays some proposed values for light level and color modification, but the light is able to understand a dim level d, where $d \in [0, 100]$ and colour in RGB format.

The expected behaviour of the system, after choosing a *Reading* context is that all lamps are turned on, dimmed to different level and set to color yellow. The underlying effect, not directly visible in the simulation, is organising devices in one virtual device. The effect of that organisation is visible, as lights apply settings predefined for a particular context.

A recipe responsible for setting a new context by the Switch is *Recipe 1* and a recipe responsible for setting a light is *Recipe 2* (shown in Figure 41). *Recipe 1* is responsible for setting a new virtual device, *Recipe 2* is managing the configuration for a lamp, both are designed to cooperate for context *Reading*. The user's request triggers *Recipe 1*, the first step of this recipe is displaying a message on GUI that a new context is being configured.

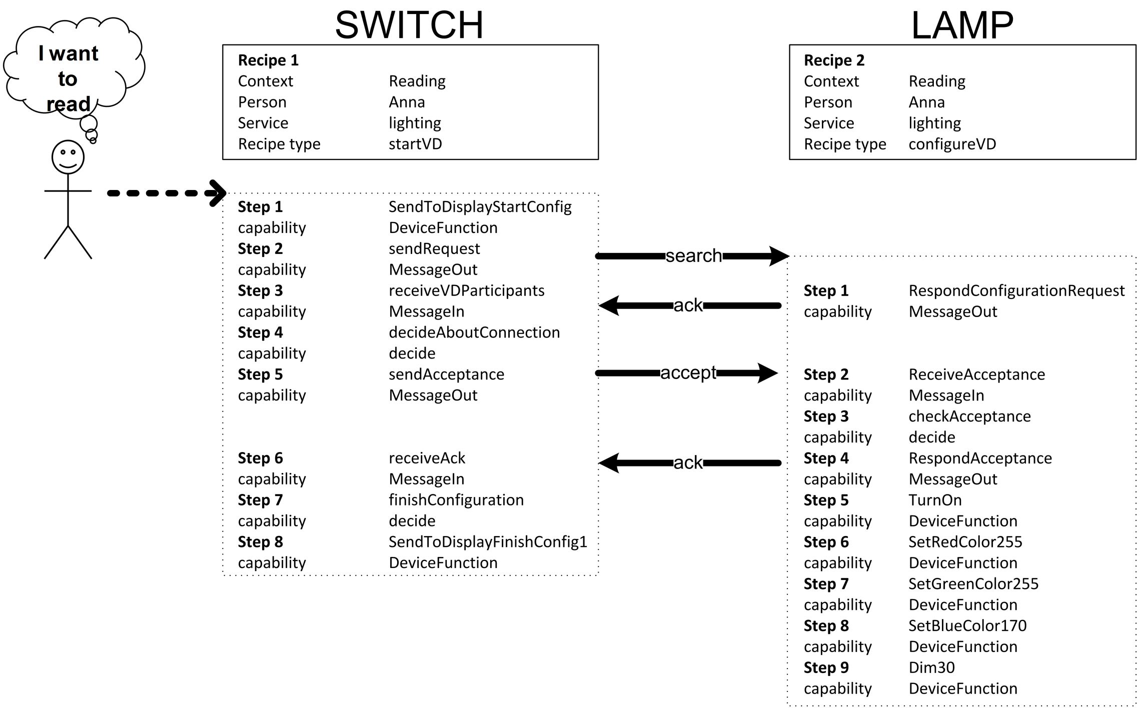


Figure 41: Devices interactions using actions from recipes.

The second step is sending a search message (Table 7) to all devices in a space to configure context *Reading*. All devices receive a search message and search their knowledgebase for context *Reading* for user *Anna*. In this example a lamp found an appropriate recipe for this context, therefore *Recipe 2* is started. The first step in *Recipe 2* is responsible for sending an acknowledgement message back to the Switch. In the mean time the Switch is ready to receive messages. When a switch receives a message and a timer triggers that the time to wait for the input is finished, the Switch proceeds to step 4 and checks received messages. The switch checks all messages and determines what devices are being accepted.

MessageType	0
SenderId	18
VirtualDeviceId	100
MessageId	1
RequestType	configureVD
Service	Lighting
Context	Reading
Person	Anna
Priority	0

Table 7: Search message for context *Reading*.

In the mean time the Lamp waits for an input and if any message arrives, step 1 is executed. The Switch continues to step 5 and sends all acceptances. The next step in the Switch is to wait for an acknowledgement from accepted devices. The Lamp receives acceptance, sends an acknowledgement and continues to step 4, next steps in *Recipe 2* turns on the light, change its color and dim level. After completing step 9 in *Recipe 2* the Lamp is finishing the recipe and setting the configuration state to *engaged*. After receiving an acknowledgement message, the Switch is again deciding if the configuration should be continued in step 7, if so it proceeds to step 8 and displays information about the current context on a screen and finishes the recipe. Both devices are designed to proceed with steps in the recipe if it is possible, aborting a step is a sign that establishing of a virtual device was not successful. Devices process recipes independently, rarely waiting for some time, specified by the recipe, and proceeding to a new step.

The initial state of the system in this experiment is presented in Figure 42A The event of pressing a button in the Switch GUI triggers running recipes in devices in a space, the result is establishing a new virtual device and setting lamps to a predefined dim level and color. The result of this event is visualised by the simulation and presented in Figure 42B.

The simulation grid consist of 16 lamps represented by circles and they are

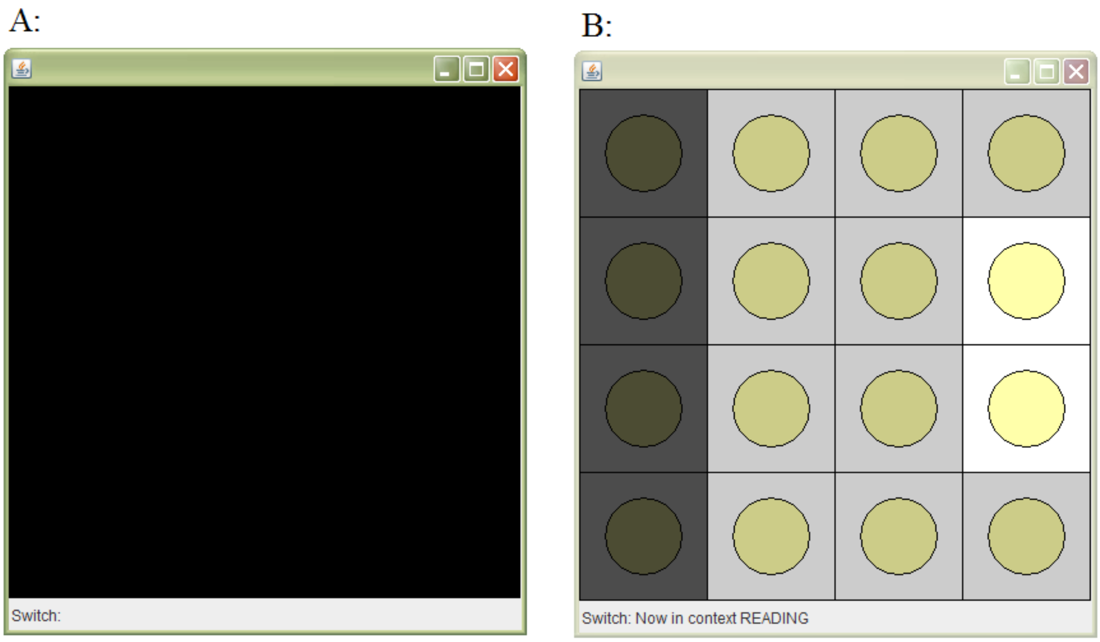


Figure 42: *Simulation of the space lightening in Experiment One, initial state.*

illuminating the floor represented by 16 squares. In context *Reading* lamps are advised by their recipes to set dim level to a different value. The values are predefined and determined by the location of the desk in a space, as depicted in the experiment scenario in Figure 14 (see Section 6).

As the context *Reading* was successfully set in the space, the lamps are controllable, by listening for commands for their virtual device. The event of pressing button *Green* in GUI (Figure 40) is expected to trigger a behaviour in lamps: change of color from yellow to green, with dim level saved from the previous setting.

The control message can be issued at any time from any device in the virtual device. Once a device finishes a configuration, with state saved to *engaged* any control message can be accepted. The recipe responsible for processing control messages in a lamp is called *Recipe 3* and is presented in Figure 43.

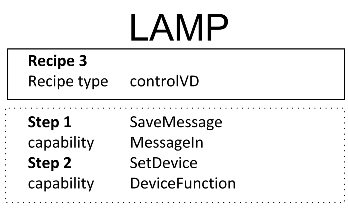


Figure 43: Simplified *Recipe3* for Lamp.

Recipe 3 is first saves the message from the input in step 1 and continues to step 2 to influence the device functionality. After this step the recipe is finished.

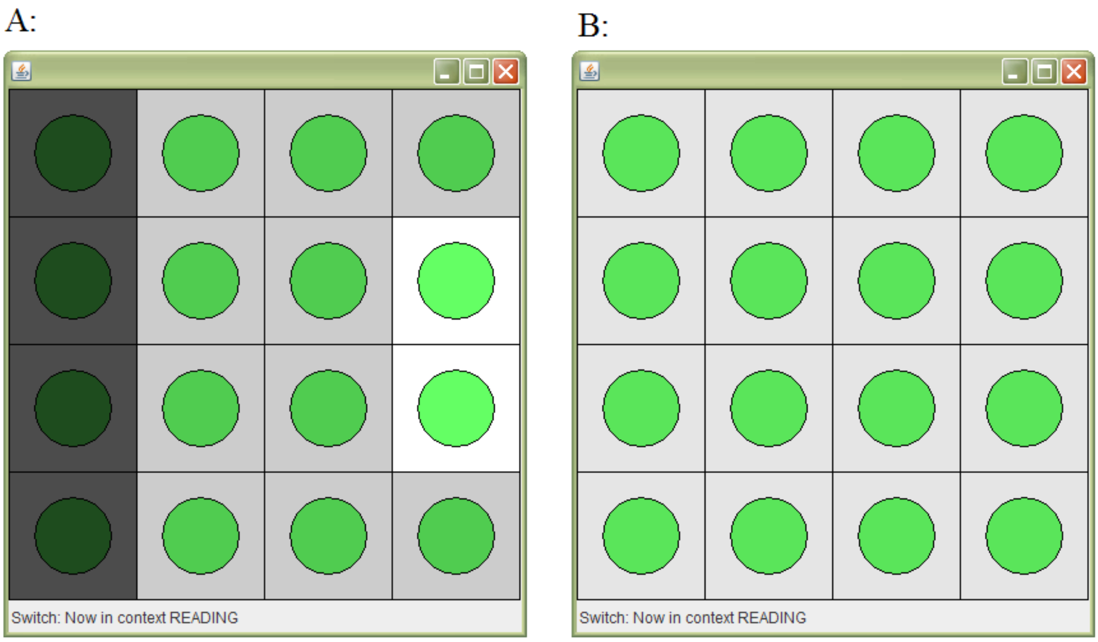


Figure 44: Simulation of the space lightening in Experiment One, control state.

The effect of running the *Recipe 3* in all lamps is shown in Figure 44A. It results in changing only the color of lamps, as only commands influencing color were used in the control message. Similarly the event of pressing button representing dim level with value 90 sets only the dim level of lights participating in the virtual device (Figure 44B).

The functional tests in the Experiment one finished with results that are expected from the set-up. The simulation runs the same implementation of the engine for all devices. The behaviour of devices is determined by the knowledgebase not by the engine implementation. The engine is flexible enough to interpret behaviour of different devices: switch and lamps. This experiment also shows that the engine can be scaled to serve different devices and interpret knowledgebases of different devices. The proposed scenario is using only some capabilities from the engine (MessageIn, MessageOut, decide and DeviceFunction), therefore the this subset of the engine's capabilities is sufficient to perform applying predefined settings and simple control of lights in a space. The engine allows devices to perform service discovery, self configuration and simple context-awareness mechanisms.

9.4 Experiment Two

Scenario two is designed to react to an environmental context of the space as well as user requirements. The space consists of 16 lamps, switch with a GUI, outdoor and indoor light sensors, and a mirror that reflect the light from outside into the room. The set-up functionality is to keep light in a level defined by context *Reading* and adapt to environmental conditions. In this scenario a mirror is added to compete with lights to provide illumination for the space (Figure 14, see Section 6).

The expected behaviour of the system, after choosing a *Reading* context is that all lamps are turned on, set to color yellow and try to maintain a level of 500 lux. The underlying effect, not directly observable in the simulation, is organising the devices into virtual devices. There are nineteen virtual devices in this scenario, as presented in Figure45. The main virtual device is VD1 that is setting context *Reading* and also triggering creation of all other virtual devices. The mirror is participating in two additional virtual devices (VD2 and VD19) to properly serve in VD1, therefore running and responding to three virtual devices at once. The expected behaviour is therefore establishing all virtual devices that are visible by lights and mirror changing their behaviour according to the environmental conditions.

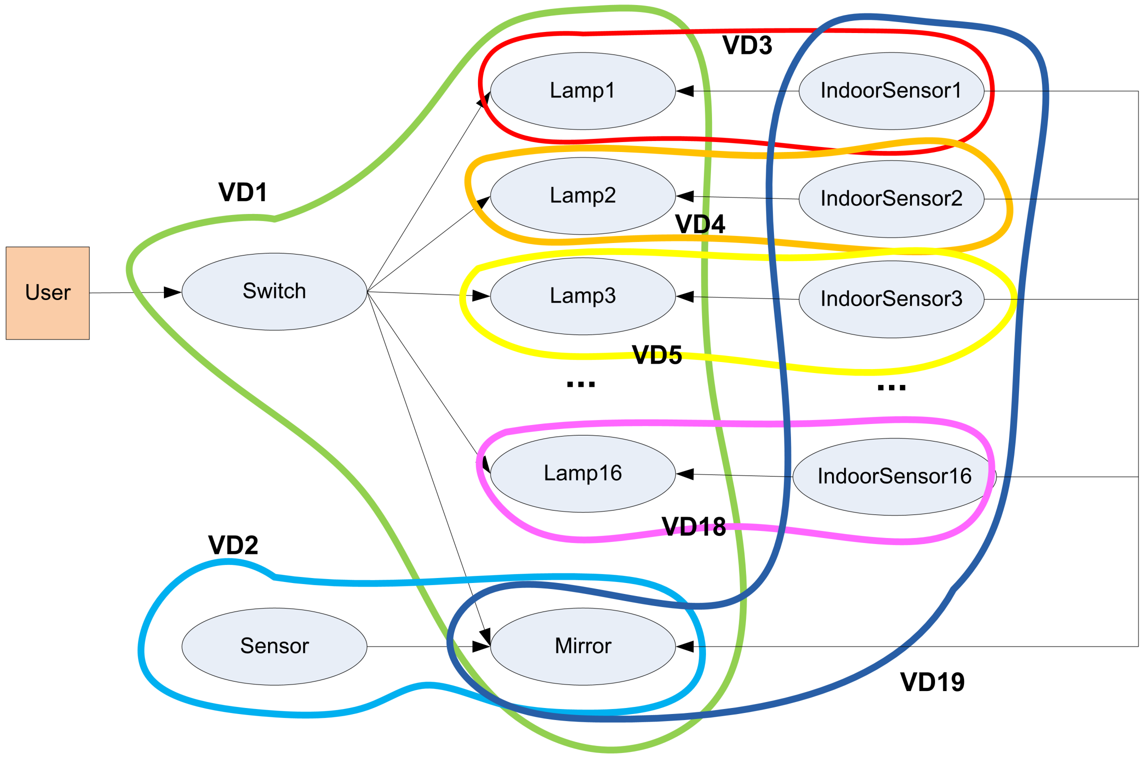


Figure 45: Virtual devices in Experiment Two.

In this experiment there are three types of interactions. First is the establishing virtual devices, the second is triggering new virtual devices depending on a situation, the third is processing data distributed in a virtual device. There are two types of distributed data: control data that propagates commands using command messages (see Section 5.4.6) and raw data equipped with data type, taken from the ontology, and a value that is numeric. The ontology is expected to define data, so the types can be distinguished. The device just uses the ontology from its own domain to interpret data. If the data is not understood by the device, the message is dropped. If the data is not of a common type, e.g. temperature, the recipe interpreting it for a particular context is responsible for properly interpreting the data.

Virtual devices in this scenario are divided into three categories depending on the virtual device purpose. The virtual device between the switch, lamps and the mirror is established to be able to send a new context for lighting, in this example the context is *Reading*. An example of a virtual device of this type is VD1 (Figure 45). The next type of a virtual device is data focused, lights and the mirror organise in a sub-networks with light sensors to gather light intensity data inside of the space. All devices providing light need this information in order to keep the light level in the space in the expected range. Therefore all lights and the mirror need to start a virtual device with light sensors located in their own space. Virtual devices VD3-VD19 (Figure 45) are this type of virtual devices. The last virtual device type VD2 is designed to gather light intensity data from outside of the room, therefore the light intensity that can be used by the mirror to illuminate the room.This information is crucial for for the mirror to determine how much light it can offer to a smart space. The mirror cannot work properly without this virtual device, therefore the VD2 uses the Poke-wait mechanism explained in Section 8.3.3.

First the virtual device targeted to establishing a new context for lighting in the space, namely VD1 in Figure 45, is explained further with an example of an interaction between the switch, lamp1 and the mirror. As VD1 triggers VD3-VD19 and VD2 (Figure 45)to gather data about light intensity indoors and outdoors respectively, this section describes the interactions and shows recipes run in VD3 and VD2 (Figure 45).

9.4.1 Triggering internal behaviour

In Scenario One only one recipe was used to configure a virtual device and set a new context. Scenario Two is more complicated, therefore the recipe running in the mirror was divided into four parts (Figure 46).

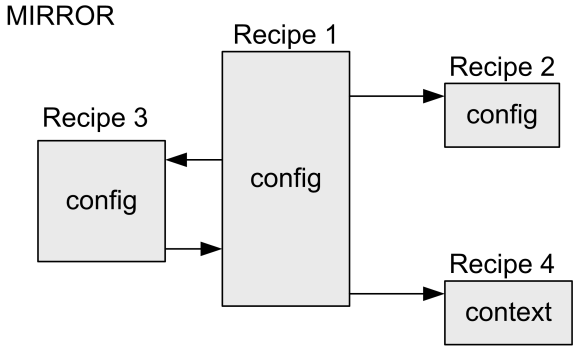


Figure 46: Mirror's recipes for VD1, VD2 and VD19 in Experiment Two.

As presented in Figure 46, *Recipe 1* is responsible for managing VD1 presented in Figure 45 and running *Recipe 2, Recipe 3* and *Rrecipe 4*. *Recipe 2* is responsible for starting a new virtual device with the indoor light sensors (VD19 in Figure 45). The *Recipe 3* manages creation of a new virtual device with an outdoor light intensity sensor (VD2 in Figure 45). *Recipes 1,2* and *3* are very general and have to be run a priori of setting the context, to ensure that devices work properly and have all resources that they need. After running *Recipes 1,2* and *3* Mirror is ready participate in a new context therefore *Recipe 4* can be run. *Recipe 4* is specially designed for context *Reading* and is only run when all configuration is successful in *Recipe 1*. *Recipe 1* triggers *Recipe 4* and *Recipe 4* usually sets some values in the STKB or influences the device functionality.

The detailed recipes for the mirror are presented in Figure 47. *Recipe 1* is run when the mirror gets a signal to configure a new context. In *Recipe 1*, *step1* is designed to send an acknowledgement message to the device starting the virtual device, next in *step2* the device issues an internal message to start a virtual device with an indoor sensor, which triggers *Recipe 2*, from this moment *Recipe 2* runs in parallel with *Recipe 1*. In *Recipe 1* in *step3* another internal action is triggered, starting a virtual device with outdoor sensor, therefore *Recipe 3* is run in parallel with *Recipe 1* and *Recipe 2*. When *Recipe 1* reaches *step6* it is forced to wait for

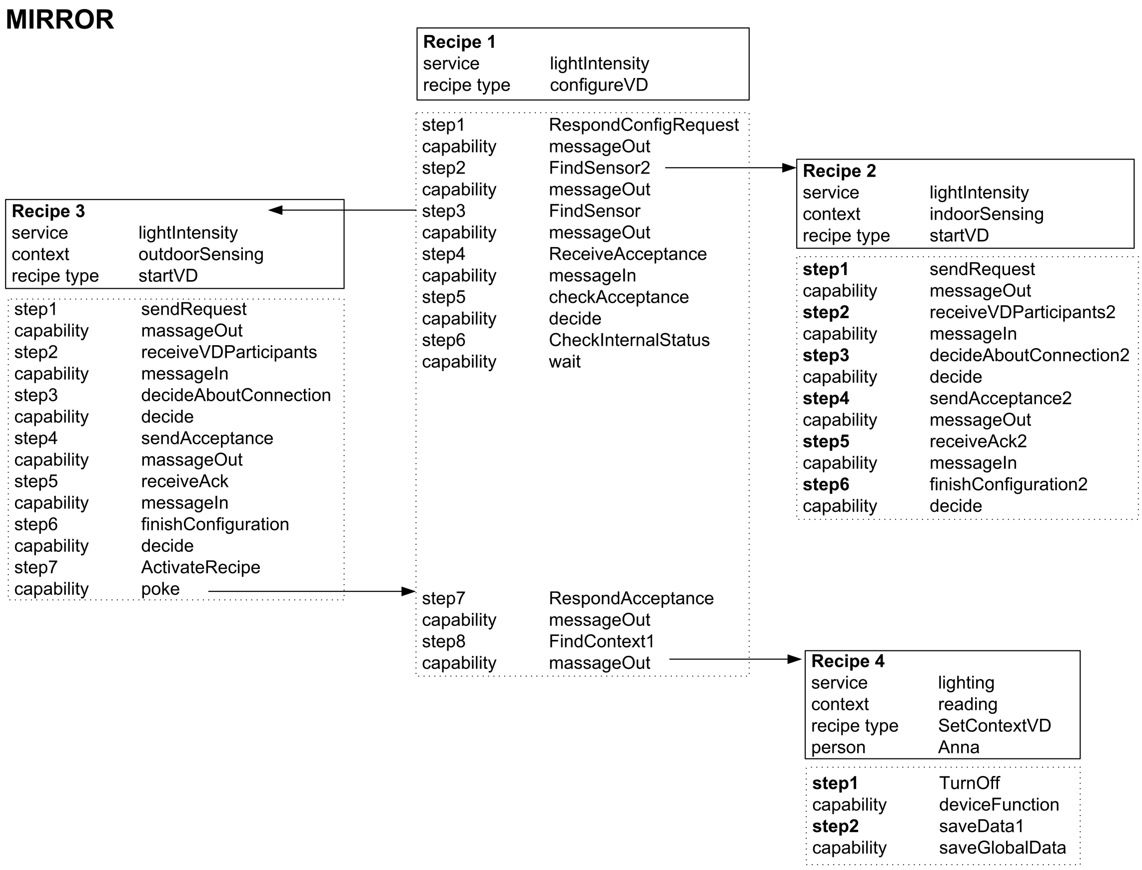


Figure 47: Detailed Mirror's recipes for VD1, VD2 and VD19 in Experiment Two.

a signal from another recipe, in this case *Recipe 3*. As use of the outdoor sensor is mandatory for the mirror, the main configuration recipe *Recipe 1* cannot continue without acknowledgement (called here poke) that this configuration was successful. When *Recipe 3* reaches *step 7* it sends a signal to Poke mechanism, that reactivate *Recipe 1*. When *Recipe 1* reaches *step8* it activates *Recipe 4* that sets the context for the space. *Recipe 2* creates a bond between the mirror and the indoor sensors and is not necessary for the mirror to function, therefore *Recipe 1* does not wait for the outcome of this recipe.

In this scenario the process of establishing the new virtual devices was divided into two parts: initial configuration independent of the context but dependent of the service (*Recipe 1,2* and *3* in Figure 47), and specific configuration that is context dependent (*Recipe 4* in Figure 47).

9.4.2 Data processing

Once virtual devices VD2-VD19 (Figure 45) are established sensors start sending data, therefore all lighting elements need recipes to process data. In the case of the mirror knowledgebase two different recipes to process indoor and outdoor light intensity data are needed, although data from sensors is of the same type, virtual devices VD2 and VD19 are different and depending on the destination of the data, it is processed differently. The data for the outdoor sensor is used determine how much light the mirror can offer to the indoor space. This value is saved in STKB to be used when interpreting data from the indoor light intensity sensor. The recipe to process indoor light intensity data is labelled as *Recipe 5* and presented in Figure 48.

Recipe 5	
service	lightIntensity
context	outdoorSensing
recipe type	processData

step1	calculateData1
capability	calculate
step2	saveData1
capability	saveGlobalData
step3	calculateData2
capability	calculate
step4	SkipSteps1
capability	skip
step5	SetDevice1
capability	deviceFunction

Figure 48: Recipe interpreting indoor light intensity values for the mirror.

The recipe is triggered when new data for context *outdoorSensing* arrives and if the data is of type *lightIntensity* (Figure 48). *Step1* takes the sensor value and divides it by 10, this is an operation that calculates the maximum dim level the mirror can provide. Next in *step2* the value of maximum dim is saved for future reference. *Step3* is used to determine if the dim level of the device is greater than the maximum dim calculated in the previous step. Next *step4* is designed to make a conditional choice based on the result from the previous calculation in *step3*, in case of true the recipe is continued to *step5*, in case of false the recipe is finished and *step5* is skipped. In *step5* the value of dim for the device is overwritten with the maximum dim. *Recipe 5*, in five easy steps, is dealing with incoming data, this includes saving data and checking if the device needs to be updated.

The recipe to interpret indoor light intensity data is much more complicated and presented in Appendix B in Figure 55. This recipe uses 24 steps to interpret light intensity and follow an algorithm developed as a part of this experiment to enable emergent behaviour for energy saving presented in Section 9.5.

The presented recipes have been designed for this specific scenario, but it is easy to deduce how to implement recipes that are requesting other resources than light intensity. Scenario Two shows an example of interactions between devices and also internal interactions enabling a device to trigger behaviours internally.

Experiment Two was performed to measure the levels of complexity that can be applied to the system. In comparison to Experiment One, the device is able to run internal actions, interpret data differently depending on the device and the virtual devices in which it is participating, calculate data, make choices depending on calculation result (skip mechanism presented in Sections 7.3.2 and 8.4.7), implement emergent behaviour with use of the same KBIE as in Experiment One.

The experiment also shows the further scalability of the system, as the same implementation can also serve mirror and sensors by interpreting especially built knowledgebases.

9.5 Emergent Behaviour of Components Providing Light in Experiment Two

In Scenario Two light and other light sources are programmed to react on a sensor value and adapt its illumination level. The ideal light intensity in a space is provided by a recipe in a device knowledgebase. The ideal value is provided with a margin, set around this value, to avoid infinite light change in a margin very close to the ideal value. In this section an emergent behaviour algorithm is presented and evaluated.

9.5.1 The Used Dimming Scheme

For the purpose of Experiment Two a dimming scheme is used with the emergent behaviour algorithm. In Scenario Two all the light sources are programmed to react on a sensor value and adapt its illumination level. The ideal light intensity in a space is provided by the user and is programmed into individual devices. The ideal value is provided with a margin, that determines a range around this value where the light changes are not noticeable by the user. Therefore the lights do not try to compensate any more while light is in the margin. This is done to avoid lights changing continuously when the value received from a sensor is in a margin very close to the ideal value.

The light intensity in an environment is divided into three groups: above agreed range, inside of the range and below it. If we let v be an ideal light intensity value in an environment, $v > 0$, m be a margin and i be actual intensity of the environment, then the value of i is in the agreed range if $i \in (v - m, v + m)$ provided $m < v$ and $m > 0$, the margin around ideal intensity level can be depicted in Figure 49. If we let t be time and i be intensity of the environment, then function $f : t \to i$ is a function of intensity at a given time.

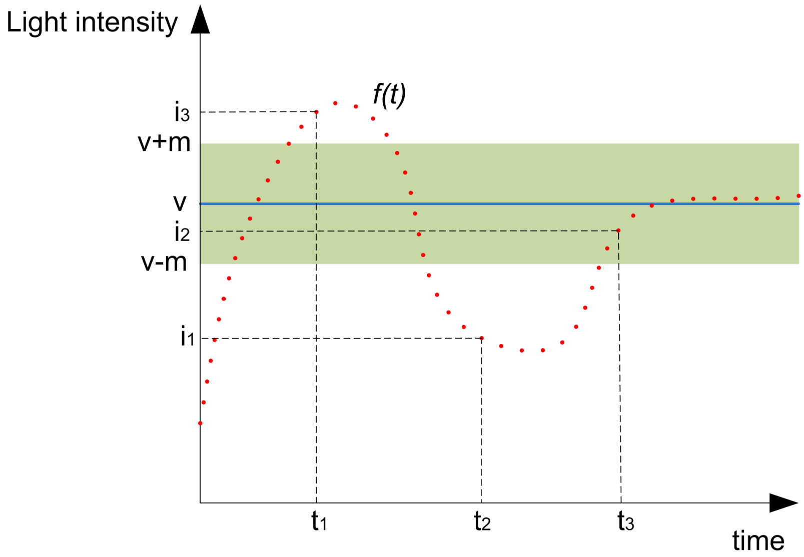


Figure 49: Ideal environment intensity value and actual sensor value over time.

The scheme that regulates light intensity i in a space takes a value from a sensor at a particular time and if the value is above the specified range, therefore $i > v+m$, the light is decreased, if the value is below the range, therefore $i < v - m$, the light is increased. If the value i is inside of the range, there is no action taken.

Functions to decrease or increase the dim level of light produced by the device are independent of actual value of intensity in a space. The value of the sensor might not be very reliable, therefore the factor of light increase or decrease is not a function of light intensity i, but of the range. The dimming scheme only checks if the value belongs to any of three ranges and uses functions not based on absolute value of i, but dependent on the range, to calculate the increase or decrease factor. Therefore the device that adapts to the environment only need to know if the value is outside of the set $(v-m, v+m)$ and react depending on the situation. Let d be a dim level of a light source, where $d \in [0, 100]$ and let Δd be a factor of increase or decrease of d, where $\Delta d \in [0, 100]$. Let function $g : (d, \Delta d) \rightarrow d'$ be a function to calculate the increase of the light level and function $h : (d, \Delta d) \rightarrow d'$ be a function to decrease a dim level in a device providing lighting. Then a general algorithm to regulate light intensity in a space is as follows:

if $(i < v - m)$ **then**
  $d' = g(d, \Delta d)$;
  change dim level to d';
else
  if $(i > v + m)$ **then**
    $d' = h(d, \Delta d)$;
    change dim level to d';
  else
    drop the request;
  end if
end if

Functions g and h can be adjusted to regulate device behaviour. In a simple scenario functions g and h can be as follows:

$g(d, \Delta d) = d + \Delta d, \quad \text{where } d \leq 100 - \Delta d,$

$h(d, \Delta d) = d - \Delta d, \quad \text{where } d \geq \Delta d.$

In this scheme to adjust the dim level of the light source is decreased/increased by a constant value Δd every time $i \notin (v-m, v+m)$. By changing the value of Δd it is possible to alternate behavior of light sources. To achieve emergent behaviour from lamps and adapt to changing lighting conditions an algorithm is needed.

9.5.2 Lazy and Enthusiastic Employee Algorithm

Let's consider an example of a brick making company. The company can make bricks that have different cost price. One is made form local materials, therefore it has a lower production cost, but the number of bricks that can be made in certain time is limited, due to limited resources. The second type of brick is more expensive to produce, because it is made from imported materials, but the number of bricks that can be made, during a certain period of time, is not limited. The company has a pool of employees,they are divided into lazy and enthusiastic employees. A lazy worker is not very conducive to work, on the other hand the enthusiastic employee is always happy to respond to manager's request. However both lazy and enthusiastic employees make bricks at the same rate once they start working at full capacity. A manager shouts a request to produce more or less bricks and all free employees try to fulfil the request. The manager has no control over choosing what type of bricks are used.

For optimal functioning of the factory the issues that need to be addressed are:

- Is it possible to make this brick making company reliable, as long as there are employees to work?
- How to minimize the production cost by using available employees and material?

Because the manager has no control over choosing a brick type that is being made, the algorithm to minimize production cost has to be deployed by the employees.

Let's assign all lazy employees to production of expensive bricks and very enthusiastic employees to cheap bricks production. When the request to increase bricks production arrives all available employees will respond, but enthusiastic employees will start working immediately and try to increase brick production rapidly. Lazy employees, on the other hand, will try to avoid working, therefore they will wait for a while and then will increase bricks production by a small factor. If the production of cheap bricks is enough to fulfil a request, expensive bricks are not produced or only small number of those bricks is produced. If the inexpensive production is not enough, then the lazy worker will eventually boost production of expensive bricks and take over the remaining percentage of the delivery.

When a request to decrease the production comes, the lazy workers quickly recognize opportunity to work less and are very happy to decrease the production. Enthusiastic employees, on the other hand, are very happy to work, so are not that inclined to decrease their workload. Enthusiastic employees will wait for a while for the situation to stabilize and, if it is necessary, they will decrease production only slightly.

The first reaction of both lazy and enthusiastic employees is significant for minimizing cost of production. Because the brick company has to be reliable, eventually lazy workers will have to work hard if the production of cheap bricks is not enough to fulfil the request. Similarly, the enthusiastic employees will have to work less if the production exceeds the required amount.

Another issue for emergent systems is a feedback loop problem. If, on a request, all employees are equally vigorous in changing their behaviour, the behaviour of the system alters very quickly and is proportional to the number of the employees answering a request. Therefore if employees are happy to answer a request to increase the production all at the same time, this behaviour will be escalated and trigger another request to decrease the production, that can lead to an infinite feedback loop. Introducing the Lazy and Enthusiastic Employee algorithm slows down the reaction of some parts of the factory, which reduces the rapid increase or decrease of production. The feedback loop can also appear, but it will truncate,as the reaction to decreasing and increasing production is not the same, the employees will eventually adapt to the situation and the loop will be broken.

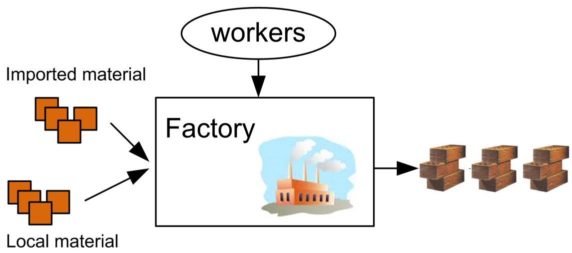


Figure 50: Factory scheme.

As shown in Figure 50 the factory is designed to supply a stable flow of bricks and react to request to increase and decrease production without any central control.

The Lazy and Enthusiastic Employee algorithm can be used to sustain a stable light level in a room with many autonomous light sources, while saving electricity. If we assume that light bulbs are lazy employees, as they need electricity to work, which is "expensive", then the emergent behaviour is to decrease light bulb use.

Whereas, the Mirror using sunlight is "cheap", is assigned to be an enthusiastic employee and it is expected to give as much light as is needed and possible. This way it is possible to use the algorithm to save energy and ensure stable light level in the space as long as all light sources can function properly.

9.5.3 Adjusting the algorithm

The main goal of the system developed for the Scenario Two is to sustain user defined light intensity in a space while maintaining low energy use, using as much natural light as possible. There are several factors that can be adjusted to achieve better results when using the lazy and enthusiastic employee algorithm: dimming function, margin's range size and skipping. An experiment was performed to determine what set of dimming, margin size and skipping parameters allows the system to adapt correctly and quickly that is measured by energy use in the space. The experiment also shows how changing parameters of the algorithm influences the behaviour of the system. The chosen parameters were tested in a controlled environment of Scenario Two, the experiment presented in this section is called Experiment 2.1.

Dimming. The first parameter of the algorithm is a dimming function. The dimming function helps varying light and mirror behaviour. As mentioned in the algorithm description lights perform a different behaviour when there is not enough light and react differently when there is to much light in space, therefore functions for dimming up and down are different. The three dimming functions tested in the Experiment 2.1 for both light and mirror are f_a,f_b and f_c, where:

- f_{x_1} is used for dimming up the light,
- f_{x_2} is used for dimming down the light,
- f_{x_3} is used for dimming up the mirror,
- f_{x_4} is used for dimming down the mirror.

Function f_a uses constant value of Δd:

$f_{a_1}(\Delta d) = 1,$

$f_{a_2}(\Delta d) = 3,$

$f_{a_3}(\Delta d) = 3,$

$f_{a_4}(\Delta d) = 1.$

In this experiment Δd is constant, but values for dimming up and dimming down are different for light and mirror.

Function f_b uses a linear function to calculate percentage of Δd being added or subtracted. *Step* is a value that is auto incremented every time the function is executed, $step \in [0, 10]$, in this experiment *step* in incremented with value 0.2:

$$f_{b_1}(\Delta d) = \Delta d \cdot (10 \cdot step)/100,$$

$$f_{b_2}(\Delta d) = \Delta d \cdot (-10 \cdot step + 100)/100,$$

$$f_{b_3}(\Delta d) = \Delta d \cdot (-10 \cdot step + 100)/100,$$

$$f_{b_4}(\Delta d) = \Delta d \cdot (10 \cdot step)/100.$$

Function f_c uses a curve based on x^4 and x^3 to calculate percentage of Δd being added or subtracted, in this experiment *step* in incremented with value 0.2:

$$f_{c_1}(\Delta d) = \Delta d \cdot (step^4/100)/100,$$

$$f_{c_2}(\Delta d) = \Delta d \cdot ((step - 10)^4/100)/100,$$

$$f_{c_3}(\Delta d) = \Delta d \cdot ((step - 10)^3/10 + 100)/100,$$

$$f_{c_4}(\Delta d) = \Delta d \cdot (100 - (step)^3/10)/100.$$

Representive curves used in functions f_b and f_c are presented in Figure 51.

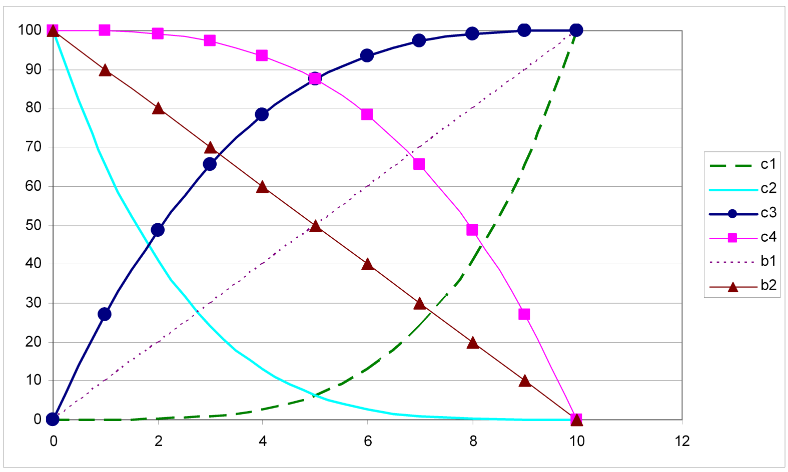


Figure 51: Dimming functions used in Experiment 2.1.

The Experiment 2.1 measures the influence of the dimming function f on energy usage in a space.

Margin size and location. The second factor that we focus on is size and location of margins that define the range of reaction in the system. The ideal intensity, defined by user, is 500 lux. There are three margins used to test the algorithm in Experiment 2.1 margin r_a, r_b and r_c.

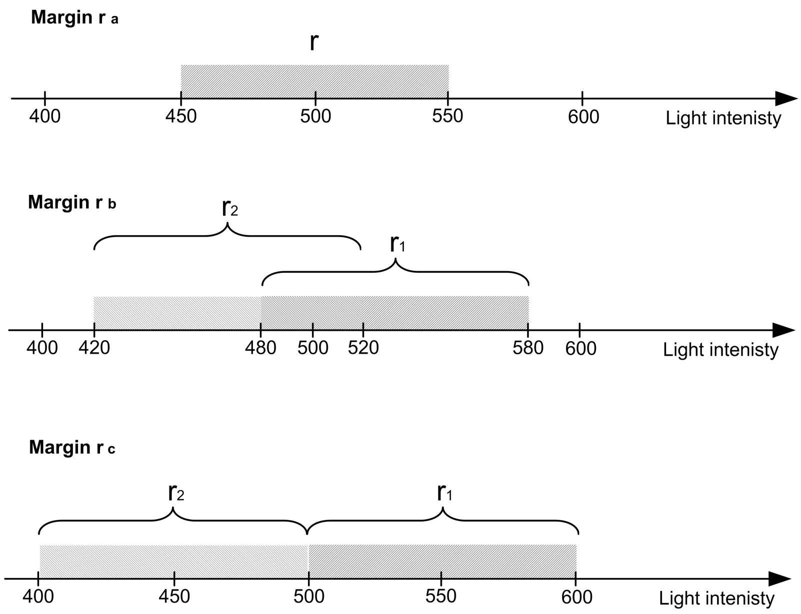


Figure 52: Regions used in Experiment 2.1.

Margin r_a is described by the set r. Both light and mirror define the same margin $m = 50$, therefore light and mirror both react on the sensor value in the same range (Figure 52). Therefore the range for both light sources is $r = [450, 550]$.

Margin r_b is described by the sets r_1 and r_2. Light and mirror uses different lower and upper margins, therefore the region is not symmetric with respect to the ideal intensity value. Lights use region r_1 and the mirror uses r_2 (Figure 52). Where r_1 and r_2 are:

$r_1 = [480, 580]$,

$r_2 = [420, 520]$.

Margin r_c is described by the sets r_3 and r_4. In this experiment margins are further adjusted to alternate behaviour of light and mirror, in this case light and mirror have excluding regions.

$r_3 = [400, 500]$,

$r_4 = (500, 600]$.

Skipping. The last experiment is designed to alternate behaviour of a lamp by ignoring part of requests from a sensor to slow down its reaction. In the Experiment

2.1 a solution without skipping s_a and with skipping s_b are presented for comparison.

Skipping s_a: the lamp accepts and analyse all the data received from sensors. Skipping s_b: the lamp is designed to accept only 2/3 of requests, while mirror accepts all requests from sensors.

9.5.4 Results and Analysis

The experiment was run in the simulation with the same environmental conditions input. Every combination of parameters of the algorithm generates an energy usage output. The energy usage is influenced by speed of adapting to the environmental conditions.

The simulation was run for 60 seconds with identical input for all experiments. The external light intensity of the environment was changed over time according to Figure 53.

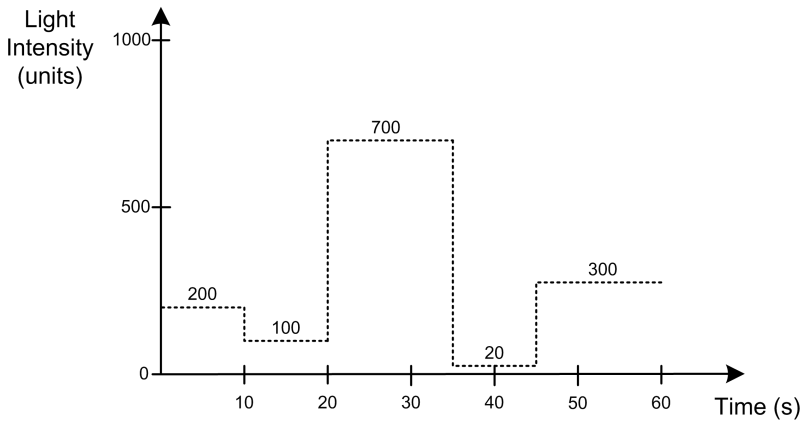


Figure 53: Input data for light intensity outside in the Experiment 2.1.

The results of the Experiment 2.1 are presented in Table 8. The best performance in achieving a goal of saving energy was for parameters Dimming f_c, Margin r_c and Skipping s_b.

This experiment shows how with use of different parameters the behaviour of the system can be adjusted. The parameters of the algorithm that helped the system use the least energy are not optimal for the algorithm, just the best from the selected set of parameters. As parameters of the algorithm can be directly changed in the knowledgebase it is easy to adjust the behaviour of lamps and mirror in the system.

The complexity of the presented system can be increased by implementing more complicated algorithms than the Lazy and Enthusiastic Employee Algorithm. The

X	X and Skipping s_a	X and Skipping s_b
Dimming f_a and Margin r_a	29945.04	27648.51
Dimming f_a and Margin r_b	28295.07	27391.13
Dimming f_a and Margin r_c	18425.26	18053.12
Dimming f_b and Margin r_a	25323.55	25003.84
Dimming f_b and Margin r_b	23475.96	23221.00
Dimming f_b and Margin r_c	17901.66	17051.45
Dimming f_c and Margin r_a	22517.36	21987.27
Dimming f_c and Margin r_b	21994.54	20826.87
Dimming f_c and Margin r_c	16145.03	16019.46

Table 8: The comparison of the Experiment 2.1 with different sets of parameters for the Lazy and Enthusiastic Employee Algorithm.

set of capabilities in the KBIE is capable of triggering simple to complicated behaviours within the domain of designed devices.

9.6 Experiment Three

Experiment Three is based on Scenario Three (Section 6.2.3) that demonstrates learning capability of a distributed system by use of recipe mobility. As presented in the Initial Experiment described in Section 9.1 bringing a new device into the space affects some existing devices. In this scenario a new device, a projector, is introduced to the system. This device also brings a new context *Projecting* for the lighting system. If the existing devices can be used in the new context, the projector need to inform devices how they are to be used. The projector knows the behaviour expected from light sources, therefore it can produce a recipe intended for light sources. With this mechanism it is possible to introduce forward compatibility in the system, therefore the existing system can learn how to co-operate with new devices. A new device can co-operate with the existing system as the underlying interpretation engine stays the same (KBIE), ensuring backward compatibility. This enables ease of integrating new components with the system and ease of reconfiguration and maintenance. The integration is done automatically and without any manual maintenance mechanisms.

When a new device, the projector, enters the space it is responsible for finding out if there are any devices that support a context that it wants to start in a space. To find out if there is any devices that can participate in the new context, but are not aware of it, the projector sends an update message (see Section 5.4.9) to group all devices that need to find out about a new context and can support the requested service. An example of an update message for context *Projecting* is presented in Table 9.

MessageType	8
SenderId	21
VirtualDeviceId	150
MessageId	5
Context	Projecting
Service	Lighting

Table 9: Update message for context *Projecting*.

Any device that gets the message is checking if it supports the requested service *lighting* and does it understand context *Projecting*. If the device supports the service but is not aware of the context, it is grouped in a virtual device, so an appropriate recipe can be sent. This virtual device is established by use of *RecipeR4* present in the lamp knowledgebase (see Appendix A, listing 730-771), this recipe type is *getUpgraded*. The first step *RecipeR4S1* of this recipe is sending an acknowledgement message if the device does not know the context, but is able to provide the service. In the next step *RecipeR4S2* the device uses the *receiveTransmission* capability (see Section 8.4.12) that changes the configuration state to *transmission* that enables the device to receive transmit messages (see Section 5.4.10). Once the transmission mode is set a device is ready to receive recipes and save them in the LTKB. In the simulation implementation a simple streaming mechanism is applied to send and receive data triple by triple. An example of a transmit message sent by the projector to lamp is presented in Table 10.

MessageType	9
SenderId	21
VirtualDeviceId	150
MessageId	39
TransmissionSize	53
TripleNumber	23
Subject	Recipe100S8
Predicate	is-a
Object	step

Table 10: Transmit message for learning new context *Projecting*.

Every triple is retrieved from the message and saved to the LTKB. When the transmission is complete, so $TransmissionSize = TripleNumber + 1$, the recipe is continiued *RecipeR4* to step *RecipeR4S3* (Appendix A, listing 758-771) where acknowledgement is sent that the transmission was successful. There is no more interactions in this virtual device so it can be destroyed. After the transmission the lamp has a new recipe called *Recipe100* as presented in Table 11.

Line no.	Subject	Predicate	Object
1901	Recipe100	is-a	recipe
1902	Recipe100	has-a	person100
1903	person100	is-a	person
1904	person100	is	Anna
1905	Recipe100	has-a	context100
1906	context100	is-a	context
1907	context100	is	presentation
1908	Recipe100	has-a	service100
1909	service100	is-a	service
1910	service100	is	lighting
1911	Recipe100	has-a	recipeType100
1912	recipeType100	is-a	recipeType
1913	recipeType100	is	SetContextVD
1914	Recipe100	has-a	Recipe100S5
1915	Recipe100S5	is-a	step
1916	Recipe100S5	has-a	TurnOff100
1917	TurnOff100	is-a	action
1918	TurnOff100	has-a	deviceFunction
1919	TurnOff100	has-a	Off100
1920	Off100	is-a	on-off
1921	Off100	is	0
1922	Recipe100S5	has-a	Recipe100S8
1923	Recipe100S8	is-a	step
1924	Recipe100S8	has-a	saveData100
1925	saveData100	is-a	action
1926	saveData100	has-a	saveGlobalData
1927	saveData100	has-a	data100
1928	data100	is-a	SimpleData
1929	data100	has-a	SimpleDataField100
1930	SimpleDataField100	is-a	SimpleDataField
1931	SimpleDataField100	has-a	nameA100
1932	nameA100	is-a	name
1933	nameA100	is	idealIntensity
1934	SimpleDataField100	has-a	valueA100
1935	valueA100	is-a	value
1936	valueA100	is	300
1937	data100	has-a	SimpleDataField200
1938	SimpleDataField200	is-a	SimpleDataField
1939	SimpleDataField200	has-a	nameA200
1940	nameA200	is-a	name
1941	nameA200	is	margin
1942	SimpleDataField200	has-a	valueA200
1943	valueA200	is-a	value
1944	valueA200	is	40
1945	data100	has-a	SimpleDataField300
1946	SimpleDataField300	is-a	SimpleDataField

Line no.	Subject	Predicate	Object
1947	SimpleDataField300	has-a	nameA300
1948	nameA300	is-a	name
1949	nameA300	is	dimDelta
1950	SimpleDataField300	has-a	valueA300
1951	valueA300	is-a	value
1952	valueA300	is	6
1953	Recipe100S8	has-a	stepStop

Table 11: Light 1 from Appendix A is appended and updated with a new recipe *Recipe100*.

With the *Recipe100* appended to the knowledgebase the device can immediately get involved with context *Preojecting* and participate in it exactly how the projector requires.

In the Experiment Three two system tests are performed. Both functional tests are run in the environment with the projector, lamps, mirror and light sensors present.

The first system test is run when the projector is activated in the space. The projector sends a request for a new context *Projecting*. All devices in the space receives the request and act upon it. The expected behaviour of the system is that lighting devices do not react to the new context, theretofore no virtual device is established and light sources are not engaged in the context.

The second system test is run with the projector entering the space. The projector first starts an update mechanism that involves sending an update message. After the update stage is finished, the projector sends a request for context *Projecting*. All devices in the space receives the request and act upon it. The expected behaviour of the system is that devices adjust their behaviour to new context. Also an expected outcome of the updating mechanism is that only devices providing lighting service are updated with new recipe. The behaviour that is not observable in the simulation is that lighting devices form a virtual device with a projector and with available sensors.

The first test in this experiment is run without an updating mechanism and all devices simply discard the search message. In this test the system is behaving as expected as devices find no recipe to be run and do not have an entry informing them of the context *Projecting*. The second test is run with the updating mechanism. As a result of this experiment all lights and the mirror change their behaviour and adjust their ideal light intensity value to 300 units. The underlying virtual devices are established which results in devices behaving as expected for this functional system test.

This simple learning technique implemented in this experiment with use of recipes is enabling the system to achieve the forward and backward compatibility. A new context can be introduced to the space and enable a different set-up and co-operation.

9.7 Results

Three experiments, following chosen scenarios were performed and their behaviours were verified with functional tests presented in this chapter. Experiment One is presenting functionality of a light controlling system enabling the setting of a simple context and manually control lights. Experiment Two extended the concept of context by adding an environmental context, enabling devices to adapt to the environment to achieve an overall goal that is using as little energy as they can and sustain the ideal intensity level set by the user. Experiment Three added a learning capability with use of recipe mobility.

All three experiments were designed to measure the levels of complexity that can be applied to the system. The same KBIE is capable of dealing with simple to fairly complicated scenarios involving adapting and learning. Three experiments were performed only by changing recipes in devices existing in the space. The experiments behaved as expected from the designed scenarios, the functional tests were successfully performed in all three experiments.

Error handling was implemented in the KBIE and different errors were tested with use of unit testing. The separate lines of code or functions were tested for the error occurrence. Types of tested errors are:

1. received data errors:
    - unknown message type,
    - unknown data type;

2. knowledgebase entries errors:

    - missing entries (e.g. no indication of the next step in a recipe),
    - inconsistency with a model.

In the case of errors with received data and messages the KBIE is expected to drop these requests and continue to the ide state as indicated in Figure 27 (see Section 8.2). In case of errors with the knowledgebase content the KBIE drops a task for example not proceeding with a configuration when one of entries in a recipe is missing or not consistent with the model.

The KBIE developed in this research is suitable to run all scenarios presented in Chapter 6 and interpret knowledgebases of five different devices: light sensor, lamp, mirror, switch and projector. Following the knowledgebase design pattern expressed as an application-specific ontology model, it is possible to implement other small devices from any domain. As the description of the device and the behaviour is expressed in the knowledgebase, the KBIE can be simple, generic and flexible in the domain of small and limited devices with simple functionalities.

10 Conclusions and further work

The research presented in this thesis has created a context aware generic Knowledgebase Interpretation Engine (see Chapter 8) that enables autonomous sensors and devices to pervasively manage smart spaces using CSP as the underlying design methodology. A knowledgebase drawn from an ontology model was designed and implemented (see Chapter 7) in an easy RDF triple format (see Section 7.1.1). The expected behaviour of device in the presented scenario from Figure 14 (Chapter 6), was presented and verified by three experiments in Chapter 9.

The theoretical background of the research area was presented in Chapters 2, 3 and 4. Describing respectively different domains that are relevant to the project, a chosen concurrency model used to simulate the environment and the outline of the various knowledge representation techniques.

The Communicating Sequential Processes (Section 3.1) model was chosen to represent the pervasive environment and the interpretation engine. The system is an asynchronous, complex and distributed environment with non-deterministic behaviour, therefore an asynchronous concurrent model was chosen to represent the environment that does not rely on a global synchronisation. The engine supporting every device consists of three main parts: KBIE, communication interface and device functionality that in the final implementation are meant to be separate pieces of hardware. In practice the KBIE is also separated into two parts: interpretation component (CU and Poke) and knowledgebase fetch, where knowledgebase fetch is likely to be implemented in parallel hardware to speed up the mechanism of retrieving data from the knowledgebase. Therefore the infrastructure supporting the device consists of four independent hardware parts that does not rely on global synchronisation, but co-operate by message passing. This is the reason the CSP model was chosen to represent both the top and bottom levels of the system as presented in Figure 2 in Chapter 3. The simulation of the system was implemented in JCSP (see Section 3.3.4 and 8.6).

The developed infrastructure for autonomous devices consisting of knowledgebase and the KBIE can be classified as a software or a middleware infrastructure. The knowledgebase can be interpreted as a software program with a set of classes (recipes), functions (steps), instructions (capabilities) and variables (instances of classes in the ontology), the KBIE can be called a compiler for this very restricted language. Once the code from the knowledgebase is interpreted it can be executed instruction by instruction in the KBIE. The presented framework is comparable to a software infrastructure and can be evaluated for its usability for pervasive computing.

10.1 Software requirements

Software requirements for pervasive computing were described in [23]. The software environment appropriate to support pervasive system must sustain application requirements such as: mobility and distribution, adaptation, interoperability, component discovery, development and deployment, scalability and context awareness.

10.1.1 Mobility and distribution

In a pervasive system mobility and distribution are natural requirements. Devices can change their location, enter or leave the space at any time. Mechanisms associated with this requirement should be deployed in software and transparently for component developers [23]. Mobility should be achieved without thinking about synchronization or data migration[23]. The services and data should be distributed and easily accessed by other devices.

In the presented architecture data, functionalities and services are distributed. The data is usually private to a device and can be sent to other devices locally, within a virtual device or to a specific device. The topology of the network is flexible and devices can come and go and if there is another device willing to take over, the virtual device can continue to work.

10.1.2 Context awareness

The pervasiveness of a computer infrastructure can be achieved by replacing input from a user with interpreting context information. The context information like activities in which the user is engaged, environmental conditions, presence of other devices can be used for adaptation and self-configuration.

In the presented architecture devices react differently in different contexts. The context can be associated with environmental conditions, presence of particular devices or people, users' intent, or periods and events associated with a calendar. Every device has its own, individual interpretation of a context, therefore all devices individually know how to behave in a particular situation and the behaviour of the system emerges from the actions of a specific component. The presented infrastructure supports context-awareness by gathering information from sensors and interpreting the data differently depending on the context.

10.1.3 Adaptation

In a highly mobile and distributed environment is dictating a dynamic nature of a pervasive environment. The infrastructure supporting such a system needs to have the ability to reconfigure and adapt to ever-changing requirements and environmental conditions.

The ability to adapt to requirements is facilitated by a simple learning mechanism, that is transferring recipes between devices. This enables device to learn behaviours from different devices in order to support novel tasks. Adaptation to changing environment is solved by individual and autonomous reaction to the event by every device. Recipes with conditional choices can dictate different behaviour depending on sensor values, the virtual devices in which a component is participating, a context or a person. The overall adaptation of the system emerges from the individual behaviour of its components (see Section 9.5).

10.1.4 Interoperability

A pervasive infrastructure is required to integrate diversity of components programmed using different languages into an infrastructure that can successfully interact and cooperate [23].

The interoperability on the semantic level can be achieved by using a flexible and updatable standard, an ontology, to provide devices with understanding concepts and relations between these concepts. The cooperation between devices on the protocol level is achieved by defining message format and exchange protocols in an unpalatable ontology. The generic KBIE can run any recipe that is constructed according to the application-specificity ontology model (see Section 7.3.1) and is represented in a simplified RDF format. The RDF formatted data can be easily translated to XML and understood by any XML parser, therefore a recipe from a device with simple hardware KBIE can be parsed to XML and understood in any device running any software variation of the interpretation component based on the same ontology. The communication component is separate from the KBIE and has to incorporate interoperability, for example by using common communication protocols.

10.1.5 Component discovery

In pervasive systems network topology can change rapidly. As a resource or service discovery is not present, devices need to individually deal with discovery.

In the presented infrastructure service discovery is performed by use of a broadcast protocol. Whenever a service is needed, a device sends a request to form a virtual device (see *search message*, Section 5.4.1) in order to fulfil the requirement. Once the virtual device is formed a task is treated locally. For example a sensor is sending it's reading data only when it is participating in a virtual device and the data is only received by other participants of this virtual device. A device can participate in many virtual devices and there are messages that the device needs to respond to, independently of engagement in other tasks (see *shout message*, Section 5.4.4).

10.1.6 Development and deployment

Components in pervasive systems are required to adapt to changing environmental conditions that requires the ability to redeploy and adapt at a runtime, without restarting devices or installing new versions of components [23]. This ability is very useful for maintenance of a large scale systems, when access to separate components is not very easy, for example sensor networks monitoring the sea bottom.

In the presented infrastructure the KBIE behaviour is dictated by the knowledgebase, therefore its possible to change or alternate a device's behaviour only by modifying its knowledgebase at a runtime. The learning capability can be executed locally, so devices to be updated can be easily chosen and modified. The generic KBIE can run a new recipe in the next cycle of the engine, without restarting.

10.1.7 Scalability

The number of devices and users in a pervasive environment is not limited. Therefore the number of interactions increases and also the number of devices increases. As a result scalability is a problem for pervasive systems [15].

In the presented system devices are autonomous, the services are distributed and cooperation between devices occurs on the service level. Devices can be grouped into virtual devices, cooperate and then finish a configuration and engage in different activities. The number of devices in this environment is not limited, and the architecture can support any number of devices. The number of virtual devices that the device can participate is also not limited. The grouping or clustering capability helps controlling devices locally. The only scalability problem can occur with connection with a broadcast nature of messages sent between devices, when many requests are issued a device needs time to process requests in the KBIE.

The software requirements for pervasive computing presented in [23] can be fulfilled by the developed infrastructure by supporting mobility and distribution, adaptation, interoperability, component discovery, development and deployment, scalability and context awareness. Therefore the presented architecture can be used as an underlining infrastructure for a pervasive environment.

10.2 Further work

The presented project is part of a bigger research programme, called the Medusas project, that is still continued at NXP Semiconductors. There are two main stems of the future focus of the Medusas project. The first is implementing a device from the system developed in this research in the C programming language, done by translating the KBIE Java code to C with as few modifications as possible and using the same knowledgebases as in the simulation. The second stream of the project is trying to improve data retrieval from a knowledgebase. Work on a parallel

Knowledgebase Fetch mechanism (see Section 8.3) is being undertaken at Technische Universiteit Eindhoven (*www.tue.nl*). The parallel version of the Knowledgebase Fetch component is being designed and implemented using a FPGA device.

10.2.1 Learning by assessing

The learning mechanism presented in Experiment Three (see Section 9.6) is fairly simple and based on the KBIE being generic and able to run any recipe built according to the model. A new device in a space can group devices and next send a recipe that enables cooperation. The mechanism to retrieve a recipe is also described in the knowledgebase. Once a new recipe is received by the device its consistency with rest of the LTKB need to be checked. First of all it needs to be constructed according to the ontology model present in the knowledgebase. There can be new data types, message formats and other data that is not represented in the engine, this data and its format can be transferred to a device together with a recipe. The learning presented in Experiment Three (Section 9.6) is not changing recipes, but adding new ones to react in a new situation and context. The next step for adapting to a new situation, not implemented in project, is learning by assessing. This mechanism enables modifying already existing recipes to adjust a device's behaviour without any external intervention. In the presented project the only self-adjustment of a behaviour of a recipe is modification of variables in STKB that are used in a recipe and together with skipping mechanism (see Section 7.3.2) one recipe can be run differently depending on input data. This mechanism is only applicable to situations that have been predicted, but is not dealing with unexpected circumstances. The learning mechanism needs to be improved in the future implementations.

10.2.2 Replacement parts

The recipe transferring mechanism (see Section 8.4.11 and 9.6) can be used in a back-up process. If a device recognises it is about to fail, it can send the content of its knowledgebase to other device in order to create a back-up of its behaviour. When a replacement devices is installed the knowledgebase can be transferred into the new device for a pervasive configuration. If the replacement device is identical with the device that has failed, the knowledgebase can be entirely overwritten. In the other case the new device needs to check what information is needed and what recipes can be discarded. The latter case needs another mechanism that checks a new knowledgebase and decides what is useful.

10.2.3 Knowledgebase Maintenance Engine

In time the knowledgebase can grow to a big size, therefore garbage collection and maintenance mechanisms are needed. A simple garbage collection mechanism was implemented in the KBIE, ensuring that when a configuration is deleted, all data associated with this configuration is also erased. As the modification of STKB is not restricted, and can be done by a recipe (see Section 8.4.6), it is possible that some data can be added and never used. The garbage collecting mechanism should delete data that is not used, for example, data that have loose ends in the ontology tree, therefore impossible to track from higher structures like configuration entries or aggregates. of add recipes adding ontology models to the LTKB can cause inconsistencies. Possible errors can be as follows: duplicated entries or whole recipes, inconsistency with the ontology model, entry name duplication, mistakes in recipes links between steps and many more. Therefore a Knowledgebase Maintenance Engine (KBME) can be implemented as a part of the new versions of the reasoning component. The KBME can be a part of KBIE or an independent unit that is reasoning about the information being added to the knowledgebase and dealing with the garbage collection.

10.2.4 GUI for creating recipes

A recipe is built according to a recipe ontology model presented in Figure 19 (see Section 7.3.1). It is possible to implement a Graphical User Interface to create new recipes. Creating recipes from scratch can be a time-consuming task, especially because of the RDF format of all the entries in the knowledgebase. An automatic recipe creation GUI enables a fast and correct way of creating recipes and can be used by device manufactures or users to create a new behaviour. A new recipe can be transported to a device using the updating mechanism explained in details in

Section 8.4.11.

References

[1] Smart Objects For Intelligent Applications. www.sofia-project.eu, 2009.

[2] Andr Bakkers, Gerald Hilderink, and Jan Broenink. A distributed real-time java system based on csp. In *Proc. Third IEEE Int. Symp. On Object Oriented Real-Time Distributed Computing ISORC'2000*, pages 400–407. IEEE, 1999.

[3] Andrew Bardsley and Doug Edwards. Compiling the language balsa to delay insensitive hardware. In *Proceedings of the IFIP TC10 WG10.5 international conference on Hardware description languages and their applications : specification, modelling, verification and synthesis of microelectronic systems: specification, modelling, verification and synthesis of microelectronic systems*, pages 89–91, London, UK, UK, 1997. Chapman & Hall, Ltd.

[4] Twan Basten, Marc Geilen, and Harmke de Groot. *Ambient Intelligence.* Springer, 2003.

[5] Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor.* Harper San Francisco, 1st edition, 1999.

[6] Ronald Bourret. XML and databases, 2005.

[7] Dr.ir. J.F. Broenink and Ir. G.H. Hilderink. A structured approach to embedded control systems implementation. In M.W. Spong, D. Repperger, and J.M.I. Zannatha, editors, *IEEE International Conference on Control Applications, 2001, CCA '01, Mexico City, Mexico*, pages 761–766. IEEE, 2001.

[8] Jan F. Broenink, Dusko S. Jovanovic, and Gerald H. Hilderink. Controlling a mechatronic set-up using real-time linux and ctc ++. In J. Amerongen van, B. Jonker, P. Regtien, and S. Stramigioli, editors, *8th Mechatronics Forum International Conference, Mechatronics 2002*, pages 1323–1331. University of Twente, 2002.

[9] buildingSMART. Industry foundation classes.

[10] Vinton Cerf, Yogen Dalal, and Carl Sunshine. Specification of internet transmission control program, 1974.

[11] Erik Christensen, Francisco Curbera, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web service definition language (WSDL), 2001.

[12] M. Scott Corson, Joseph P. Macker, and Gregory H. Cirincione. Internet-based mobile ad hoc networking. *IEEE Internet Computing*, 1999.

[13] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Pearson Education, 2005.

[14] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *In HUC 99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.

[15] Chip Elliott and Bob Heile. Self-organizing, self-healing wireless networks. *Aerospace Conference Proceedings, 2000 IEEE*, pages pp. 355–362, 2000.

[16] Dieter Fensel, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. Oil: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16:38–45, March 2001.

[17] J. Finlay and A. Dix. *An introduction to artificial intelligence*. UCL Press, 1996.

[18] Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, Ro Oltramari, and Luc Schneider. Sweetening ontologies with dolce. pages 166–181. Springer, 2002.

[19] Nikolaos Georgantas. Semantics-based interoperability for pervasive computing systems, April 2008.

[20] T. Gruber. Ontolingua: A mechanism to support portable ontologies. Technical report, Stanford University, Knowledge Systems Laboratory, 1992.

[21] John Storrs Hall. Utility fog: A universal physical substance. *NASA Conference Publication 10129*, July 1993.

[22] Sumi Helal, William Mann, Hicham El-Zabadani, Jeffrey King, Youssef Kaddoura, and Erwin Jansen. The gator tech smart house: A programmable pervasive space. *IEEE Computer Society*, pages pp.50–60, 2005.

[23] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Infrastructure for pervasive computing: Challenges. In *Workshop on Pervasive Computing INFORMATIK 01*, Vienna, 2001.

[24] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[25] C. A. R. Hoare and C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, 1985.

[26] Gerard J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[27] Mark Homewood, David May, David Shepherd, and Roger Shepherd. The ims t800 transputer. *IEEE Micro*, 7:10–26, September 1987.

[28] Ian Horrocks. Daml+ oil: a description logic for the semantic web, 2001.

[29] Dusko S. Jovanovic, Bojan Orlic, Geert K. Liet, and Jan F. Broenink. gcsp: A graphical tool for designing csp systems. In *in Communicating Process Architectures*, pages 233–251. IO Press, 2004.

[30] Soininen Juha-Pekka and Lappetelinen Antti. M3 smart environment infrastructure. In *NoTA 2009 Conference*, 2009.

[31] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for smart dust. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 271–278, New York, NY, USA, 1999. ACM.

[32] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and Michel Scholl. Rql: A declarative query language for rdf. In *The Eleventh International World Wide Web Conference (WWW'02)*, 2002.

[33] Randy H. Katz. Adaptation and mobility in wireless information systems. *IEEE Communications Magazine*, 40(5):pp. 102–114, 2002.

[34] Jon M. Kerridge, Anna Kosek, and Aly Syed. Modelling a Multi-Core Media Processor Using JCSP. In *Communicating Process Architectures 2008*, sep 2008.

[35] Jon M. Kerridge, Anna Kosek, and Aly Syed. JCSP Agents-Based Service Discovery for Pervasive Computing. In *Communicating Process Architectures 2009*, sep 2009.

[36] Latifur Khan and Yan Rao. A performance evaluation of storing XML data in relational database management systems. In *Proceedings of the 3rd international workshop on Web information and data management*, pages 31–38, Atlanta, Georgia, USA, 2001. ACM.

[37] Anna Kosek, Aly Syed, Jon Kerridge, and Alistair Armitage. A dynamic connection capability for pervasive adaptive environments using jcsp. In *The Thirty Fifth Annual Convention of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB'09)*, 2009.

[38] Anna M. Kosek, Aly A. Syed, and Jon M. Kerridge. Rdf recipes for context-aware interoperability in pervasive systems. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 1017 –1022, 2010.

[39] Shrirang Ambaji Kulkarni and G. Raghavendra Rao. Mobility model perspectives for scalability and routing protocol performances in wireless ad-hoc network. In *2008 First International Conference on Emerging Trends in Engineering and Technology*, 2008.

[40] Ora Lassila and Ralph R. Swick. Resource description framework (rdf) model and syntax specification, 1998.

[41] Jeremy M. R. Martin and S. A. Jassim. How to Design Deadlock-Free Networks Using CSP and Verification Tools – A Tutorial Introduction. In Andrè W. P. Bakkers, editor, *Proceedings of WoTUG-20: Parallel Programming and Java*, pages 326–338, mar 1997.

[42] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, and Alessandro Oltramari. WonderWeb deliverable d18, 2001.

[43] A.A. McEwan and S. Schneider. A verified development of hardware using csp/spl par/b. *Formal Methods and Models for Co-Design, ACM/IEEE International Conference on*, 0:81–82, 2006.

[44] Robin Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

[45] M. Minsky. *The society of mind*. A Touchstone book. Simon & Schuster, 1988.

[46] Robert Neches, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator, and William Swartout. Enabling Technology for Knowledge Sharing. *AI Magazine*, 12(3):36–56, August 1991.

[47] Robert Orfali, Dan Harkey, and Jeri Edwards. *The essential distributed objects survival guide*. John Wiley and Sons, Inc., 1995.

[48] Declan O'Sullivan and David Lewis. Semantically driven service interoperability for pervasive computing. In *Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*, pages 17–24, San Diego, CA, USA, 2003. ACM.

[49] Dick Pountain and David May. *A tutorial introduction to Occam programming*. McGraw-Hill, Inc., New York, NY, USA, 1987.

[50] Eric Prud'hommeaux and Andy Seaborne. (sparql query language for rdf), 2008.

[51] Awais Rashid and Gerd Kortuem. Adaptation as an aspect in pervasive computing. In *OOPSLA 2004 Workshop on Building Software for Pervasive Computing*, Vancouver, British Columbia, Canada, 2004.

[52] R. Reiter. *On closed world data bases*, pages 300–310. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[53] A. W. Roscoe. Modelling and verifying key-exchange protocols using csp and fdr. In *In 8th IEEE Computer Security Foundations Workshop*, pages 98–107. Press, 1995.

[54] A. W. Roscoe, C. A. R. Hoare, and Richard Bird. *The Theory and Practice of Concurrency*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[55] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

[56] Bill N. Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IN PROCEEDINGS OF THE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS*, pages 85–90. IEEE Computer Society, 1994.

[57] NXP Semiconductors. www.nxp.com, 2009.

[58] Richard Sharp and Robinson College. Higher-level hardware synthesis, 2002.

[59] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. Owl web ontology language guide, w3c recommendation, 2004.

[60] V Stanford, J Garofolo, O Galibert, M Michel, and C Laprun. The nist smart space and meeting room projects: signals, acquisition annotation, and metrics. *IEEE International Conference on Acoustics, Speech, and Signal Processing 2003. Proceedings. (ICASSP '03)*, 2003.

[61] Thomas Strang and Claudia Linnhoff-Popien. Service interoperability on context level in ubiquitous computing environments, 2003.

[62] Aly A. Syed, Johan Lukkien, and Roxana Frunza. An ad hoc networking architecture for pervasive systems based on distributed knowledge. In *Proceedings of Date2010, Dresden*, 2010.

[63] Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall. *Mobile Object Systems Towards the Programmable Internet: A Note on Distributed Computing*, volume Volume 1222/1997. Springer Berlin / Heidelberg, 1994.

[64] X. Wang, M. Kwiatkowska, G. Theodoropoulos, and Q. Zhang. Towards a unifying csp approach to hierarchical verification of asynchronous hardware. *Electron. Notes Theor. Comput. Sci.*, 128:231–246, May 2005.

[65] Roy Want. Sensor-driven computing comes of age. *Pervasive Computing, IEEE*, pages 4–6, 2007.

[66] Rolf H. Weber and Romana Weber. *Internet of Things*. Springer, 2010.

[67] Mark Weiser. The computer for the 21st century. *Scientific American*, September 1991.

[68] P. H. Welch and P. D. Austin. The jcsp home page. http://www.cs.ukc.ac.uk/projects/ofa/jcsp/, 1999.

[69] David C. Wood and Peter H. Welch. The kent retargetable occam compiler. In *Proceedings of the 19th world occam and transputer user group technical meeting on Parallel processing developments*, pages 143–166, Amsterdam, The Netherlands, The Netherlands, 1996. IOS Press.

A Appendix A: Knowledgebase Example

An Example of a long-term knowledgebase of a lamp is presented in the following table.

Line no.	Subject	Predicate	Object
1	lightSource	is-a	smart-object
2	LS1	is-a	lightSource
3	LS1	has-a	location1
4	location1	is-a	location
5	location1	is	1
6	LS1	has-a	id1
7	id1	is-a	id
8	id1	is	1
9	LS1	has-a	ec1
10	ec1	is-a	energyConsumption
11	ec1	is	20
12	LS1	has-a	service1
13	service1	is-a	service
14	service1	is	lighting
15	on-off	is-a	command
16	color-blue	is-a	command
17	color-red	is-a	command
18	color-green	is-a	command
19	dim	is-a	command
20	on-off	is	ON-OFF
21	color-blue	is	BLUE
22	color-red	is	RED
23	color-green	is	GREEN
24	dim	is	DIM
25	Recipe1	is-a	recipe
26	Recipe1	has-a	service1
27	Recipe1	has-a	recipeType1
28	recipeType1	is-a	recipeType
29	recipeType1	is	configureVD
30	Recipe1	has-a	Recipe1S1
31	Recipe1S1	is-a	step
32	Recipe1S1	has-a	RespondConfigRequest
33	RespondConfigRequest	is-a	action
34	RespondConfigRequest	has-a	massageOut
35	RespondConfigRequest	has-a	message1
36	message1	is-a	ackMessage
37	message1	has-a	field1
38	field1	is-a	messageField
39	field1	has-a	id2
40	id2	is-a	id
41	id2	is	5
42	field1	has-a	value2
43	value2	is-a	value
44	value2	is	0

Line no.	Subject	Predicate	Object
45	field1	has-a	name2
46	name2	is-a	name
47	name2	is	ackResponse
48	Recipe1S1	has-a	Recipe1S2
49	Recipe1S2	is-a	step
50	Recipe1S2	has-a	FindSensor
51	FindSensor	is-a	action
52	FindSensor	has-a	massageOut
53	FindSensor	has-a	message2
54	message2	is-a	internalMessage
55	message2	has-a	fieldB2
56	fieldB2	is-a	messageField
57	fieldB2	has-a	idB2
58	idB2	is-a	id
59	idB2	is	7
60	fieldB2	has-a	valueB2
61	valueB2	is-a	value
62	valueB2	is	indoorSensing
63	fieldB2	has-a	nameB2
64	nameB2	is-a	name
65	nameB2	is	Context
66	message2	has-a	fieldB1
67	fieldB1	is-a	messageField
68	fieldB1	has-a	idB1
69	idB1	is-a	id
70	idB1	is	4
71	fieldB1	has-a	valueB1
72	valueB1	is-a	value
73	valueB1	is	startVD
74	fieldB1	has-a	nameB1
75	nameB1	is-a	name
76	nameB1	is	RequestType
77	message2	has-a	fieldB3
78	fieldB3	is-a	messageField
79	fieldB3	has-a	idB3
80	idB3	is-a	id
81	idB3	is	5
82	fieldB3	has-a	valueB3
83	valueB3	is-a	value
84	valueB3	is	lightIntensity
85	fieldB3	has-a	nameB3
86	nameB3	is-a	name
87	nameB3	is	Service
88	Recipe1S2	has-a	Recipe1S3
89	Recipe1S3	is-a	step
90	Recipe1S3	has-a	ReceiveAcceptance
91	ReceiveAcceptance	is-a	action
92	ReceiveAcceptance	has-a	messageIn
93	ReceiveAcceptance	has-a	requestingDevice
94	requestingDevice	is-a	messageAggregate

Line no.	Subject	Predicate	Object
95	ReceiveAcceptance	has-a	messageA1
96	messageA1	is-a	acceptMessage
97	timer1	is-a	timer
98	timer1	is	1000
99	ReceiveAcceptance	has-a	timer1
100	Recipe1S3	has-a	Recipe1S4
101	Recipe1S4	is-a	step
102	Recipe1S4	has-a	checkAcceptance
103	checkAcceptance	is-a	action
104	checkAcceptance	has-a	decide
105	checkAcceptance	has-a	requestingDevice
106	Recipe1S4	has-a	Recipe1S6
107	Recipe1S5	is-a	step
108	Recipe1S5	has-a	CheckInternalStatus
109	CheckInternalStatus	is-a	action
110	CheckInternalStatus	has-a	wait
111	Recipe1S5	has-a	Recipe1S6
112	Recipe1S6	is-a	step
113	Recipe1S6	has-a	RespondAcceptance
114	RespondAcceptance	is-a	action
115	RespondAcceptance	has-a	massageOut
116	RespondAcceptance	has-a	requestingDevice
117	RespondAcceptance	has-a	message4
118	message4	is-a	ackMessage
119	message4	has-a	field4
120	field4	is-a	messageField
121	field4	has-a	id4
122	id4	is-a	id
123	id4	is	5
124	field4	has-a	value4
125	value4	is-a	value
126	value4	is	0
127	Recipe1S6	has-a	Recipe1S7
128	Recipe1S7	is-a	step
129	Recipe1S7	has-a	FindContextVDR1
130	FindContextVDR1	is-a	action
131	FindContextVDR1	has-a	massageOut
132	FindContextVDR1	has-a	messageR1
133	messageR1	is-a	internalMessage
134	messageR1	has-a	fieldR11
135	messageR1	has-a	fieldR12
136	fieldR12	is-a	messageField
137	fieldR12	has-a	idR12
138	idR12	is-a	id
139	idR12	is	4
140	fieldR12	has-a	valueR12
141	valueR12	is-a	value
142	valueR12	is	SetContextVD
143	fieldR12	has-a	nameR12
144	nameR12	is-a	name

Line no.	Subject	Predicate	Object
145	nameR12	is	RequestType
146	Recipe1S7	has-a	stepStop
147	RecipeR1	is-a	recipe
148	RecipeR1	has-a	person1
149	person1	is-a	person
150	person1	is	Anna
151	RecipeR1	has-a	context1
152	context1	is-a	context
153	context1	is	reading
154	RecipeR1	has-a	service1
155	RecipeR1	has-a	recipeType6
156	recipeType6	is-a	recipeType
157	recipeType6	is	SetContextVD
158	RecipeR1	has-a	RecipeR1S3
159	RecipeR1S3	is-a	step
160	RecipeR1S3	has-a	TurnOn
161	TurnOn	is-a	action
162	TurnOn	has-a	deviceFunction
163	TurnOn	has-a	turnOn1
164	turnOn1	is-a	on-off
165	turnOn1	is	1
166	RecipeR1S3	has-a	RecipeR1S4
167	RecipeR1S4	is-a	step
168	RecipeR1S4	has-a	SetRedColor255
169	SetRedColor255	is-a	action
170	SetRedColor255	has-a	deviceFunction
171	SetRedColor255	has-a	RedColor255
172	RedColor255	is-a	color-red
173	RedColor255	is	255
174	RecipeR1S4	has-a	RecipeR1S5
175	RecipeR1S5	is-a	step
176	RecipeR1S5	has-a	SetGreenColor255
177	SetGreenColor255	is-a	action
178	SetGreenColor255	has-a	deviceFunction
179	SetGreenColor255	has-a	GreenColor255
180	GreenColor255	is-a	color-green
181	GreenColor255	is	255
182	RecipeR1S5	has-a	RecipeR1S6
183	RecipeR1S6	is-a	step
184	RecipeR1S6	has-a	SetBlueColor0
185	SetBlueColor0	is-a	action
186	SetBlueColor0	has-a	deviceFunction
187	SetBlueColor0	has-a	BlueColor175
188	BlueColor175	is-a	color-blue
189	BlueColor175	is	175
190	RecipeR1S6	has-a	RecipeR1S7
191	RecipeR1S7	is-a	step
192	RecipeR1S7	has-a	SetDim80
193	SetDim80	is-a	action
194	SetDim80	has-a	deviceFunction

Line no.	Subject	Predicate	Object
195	SetDim80	has-a	dim80
196	dim80	is	100
197	dim80	is-a	dim
198	RecipeR1S7	has-a	RecipeR1S8
199	RecipeR1S8	is-a	step
200	RecipeR1S8	has-a	saveData1
201	saveData1	is-a	action
202	saveData1	has-a	saveGlobalData
203	saveData1	has-a	data1
204	data1	is-a	SimpleData
205	data1	has-a	SimpleDataField1
206	SimpleDataField1	is-a	SimpleDataField
207	SimpleDataField1	has-a	nameA1
208	nameA1	is-a	name
209	nameA1	is	idealIntensity
210	SimpleDataField1	has-a	valueA1
211	valueA1	is-a	value
212	valueA1	is	500
213	data1	has-a	SimpleDataField2
214	SimpleDataField2	is-a	SimpleDataField
215	SimpleDataField2	has-a	nameA2
216	nameA2	is-a	name
217	nameA2	is	margin
218	SimpleDataField2	has-a	valueA2
219	valueA2	is-a	value
220	valueA2	is	40
221	data1	has-a	SimpleDataField3
222	SimpleDataField3	is-a	SimpleDataField
223	SimpleDataField3	has-a	nameA3
224	nameA3	is-a	name
225	nameA3	is	dimDelta
226	SimpleDataField3	has-a	valueA3
227	valueA3	is-a	value
228	valueA3	is	6
229	RecipeR1S8	has-a	stepStop
230	recipe7	is-a	recipe
231	Recipe7	has-a	person1
232	Recipe7	has-a	service1
233	Recipe7	has-a	recipeType4
234	recipeType4	is-a	recipeType
235	recipeType4	is	controlVD
236	Recipe7	has-a	Recipe7S1
237	Recipe7S1	is-a	step
238	Recipe7S1	has-a	ReceiveMessage
239	ReceiveMessage	is-a	action
240	ReceiveMessage	has-a	messageIn
241	ReceiveMessage	has-a	InputData
242	InputData	is-a	messageAggregate
243	Recipe7S1	has-a	Recipe7S2
244	Recipe7S2	is-a	step

Line no.	Subject	Predicate	Object
245	Recipe7S2	has-a	SetDevice
246	SetDevice	is-a	action
247	SetDevice	has-a	deviceFunction
248	SetDevice	has-a	InputData
249	Recipe7S2	has-a	stepStop
250	recipe3	is-a	recipe
251	Recipe3	has-a	person1
252	Recipe3	has-a	context3
253	Recipe3	has-a	service2
254	service2	is-a	service
255	service2	is	lightIntensity
256	Recipe3	has-a	recipeType3
257	recipeType3	is-a	recipeType
258	recipeType3	is	processData
259	Recipe3	has-a	Recipe3S0
260	Recipe3S0	is-a	step
261	Recipe3S0	has-a	ReceiveDataMessage
262	ReceiveDataMessage	is-a	action
263	ReceiveDataMessage	has-a	messageIn
264	ReceiveDataMessage	has-a	InputData1
265	InputData1	is-a	messageAggregate
266	Recipe3S0	has-a	Recipe3S2
267	Recipe3S2	is-a	step
268	Recipe3S2	has-a	calculateData
269	calculateData	is-a	action
270	calculateData	has-a	calculate
271	calculateData	has-a	calculationData1
272	calculationData1	is-a	calculationData
273	calculationData1	has-a	calculationData1F1
274	calculationData1F1	is-a	calculationDataField
275	calculationData1F1	has-a	name5
276	name5	is-a	name
277	name5	is	calculationType
278	calculationData1F1	has-a	value5
279	value5	is-a	value
280	value5	is	arytmetic
281	calculationData1	has-a	calculationData1F2
282	calculationData1F2	is-a	calculationDataField
283	calculationData1F2	has-a	name6
284	name6	is-a	name
285	name6	is	attributeType
286	calculationData1F2	has-a	value6
287	value6	is-a	value
288	value6	is	integer
289	calculationData1	has-a	calculationData1F3
290	calculationData1F3	is-a	calculationDataField
291	calculationData1F3	has-a	name7
292	name7	is-a	name
293	name7	is	attributeX
294	calculationData1F3	has-a	value7

Line no.	Subject	Predicate	Object
295	value7	is-a	value
296	value7	is	idealIntensity
297	calculationData1	has-a	calculationData1F4
298	calculationData1F4	is-a	calculationDataField
299	calculationData1F4	has-a	name8
300	name8	is-a	name
301	name8	is	attributeY
302	calculationData1F4	has-a	value8
303	value8	is-a	value
304	value8	is	margin
305	calculationData1	has-a	calculationData1F5
306	calculationData1F5	is-a	calculationDataField
307	calculationData1F5	has-a	name9
308	name9	is-a	name
309	name9	is	operator
310	calculationData1F5	has-a	value9
311	value9	is-a	value
312	value9	is	-
313	calculationData1	has-a	calculationData1F6
314	calculationData1F6	is-a	calculationDataField
315	calculationData1F6	has-a	name10
316	name10	is-a	name
317	name10	is	outcome
318	calculationData1F6	has-a	value10
319	value10	is-a	value
320	value10	is	teporaryData1
321	Recipe3S2	has-a	Recipe3S3
322	Recipe3S3	is-a	step
323	Recipe3S3	has-a	calculateData2
324	calculateData2	is-a	action
325	calculateData2	has-a	calculate
326	calculateData2	has-a	InputData1
327	teporaryData1	is-a	SimpleData
328	calculateData2	has-a	teporaryData1
329	calculateData2	has-a	calculationData2
330	calculationData2	is-a	calculationData
331	calculationData2	has-a	calculationData2F1
332	calculationData2F1	is-a	calculationDataField
333	calculationData2F1	has-a	name5
334	calculationData2F1	has-a	value13
335	value13	is-a	value
336	value13	is	compare
337	calculationData2	has-a	calculationData2F2
338	calculationData2F2	is-a	calculationDataField
339	calculationData2F2	has-a	name6
340	calculationData2F2	has-a	value6
341	calculationData2	has-a	calculationData2F3
342	calculationData2F3	is-a	calculationDataField
343	calculationData2F3	has-a	name7
344	calculationData2F3	has-a	value14

Line no.	Subject	Predicate	Object
345	value14	is-a	value
346	value14	is	lightIntensityValue
347	calculationData2	has-a	calculationData2F4
348	calculationData2F4	is-a	calculationDataField
349	calculationData2F4	has-a	name8
350	calculationData2F4	has-a	value15
351	value15	is-a	value
352	value15	is	teporaryData1
353	calculationData2	has-a	calculationData2F5
354	calculationData2F5	is-a	calculationDataField
355	calculationData2F5	has-a	name9
356	calculationData2F5	has-a	value16
357	value16	is-a	value
358	value16	is	¡
359	calculationData2	has-a	calculationData2F6
360	calculationData2F6	is-a	calculationDataField
361	calculationData2F6	has-a	name10
362	calculationData2F6	has-a	valueA10
363	valueA10	is-a	value
364	valueA10	is	teporaryData2
365	teporaryData2	is-a	SimpleData
366	Recipe3S3	has-a	Recipe3S4
367	Recipe3S4	is-a	step
368	Recipe3S4	has-a	skipSteps1
369	skipSteps1	is-a	action
370	skipSteps1	has-a	skip
371	skipSteps1	has-a	teporaryData2
372	skipSteps1	has-a	SkipData1
373	SkipData1	is-a	SkipData
374	SkipData1	has-a	SkipDataField1
375	SkipDataField1	is-a	SkipDataField
376	SkipDataField1	has-a	name11
377	name11	is-a	name
378	name11	is	TRUE
379	SkipDataField1	has-a	value11
380	value11	is-a	value
381	value11	is	Recipe3S5
382	SkipData1	has-a	SkipDataField2
383	SkipDataField2	is-a	SkipDataField
384	SkipDataField2	has-a	name12
385	name12	is-a	name
386	name12	is	FALSE
387	SkipDataField2	has-a	value12
388	value12	is-a	value
389	value12	is	Recipe3S13
390	Recipe3S4	has-a	Recipe3S5
391	Recipe3S5	is-a	step
392	Recipe3S5	has-a	calculateData3
393	calculateData3	is-a	action
394	calculateData3	has-a	calculate

Line no.	Subject	Predicate	Object
395	calculateData3	has-a	calculationData3
396	calculationData3	is-a	calculationData
397	calculationData3	has-a	calculationData3F1
398	calculationData3F1	is-a	calculationDataField
399	calculationData3F1	has-a	name5
400	calculationData3F1	has-a	value17
401	value17	is-a	value
402	value17	is	function
403	calculationData3	has-a	calculationData3F3
404	calculationData3F3	is-a	calculationDataField
405	calculationData3F3	has-a	name7
406	calculationData3F3	has-a	value18
407	value18	is-a	value
408	value18	is	functionStepData
409	calculationData3	has-a	calculationData3F5
410	calculationData3F5	is-a	calculationDataField
411	calculationData3F5	has-a	name9
412	calculationData3F5	has-a	value19
413	value19	is-a	value
414	value19	is	function1
415	calculationData3	has-a	calculationData3F6
416	calculationData3F6	is-a	calculationDataField
417	calculationData3F6	has-a	name10
418	calculationData3F6	has-a	value10
419	Recipe3S5	has-a	Recipe3S6
420	Recipe3S6	is-a	step
421	Recipe3S6	has-a	calculateData4
422	calculateData4	is-a	action
423	calculateData4	has-a	calculate
424	calculateData4	has-a	calculationData4
425	calculationData4	is-a	calculationData
426	calculationData4	has-a	calculationData4F1
427	calculationData4F1	is-a	calculationDataField
428	calculationData4F1	has-a	name5
429	calculationData4F1	has-a	value20
430	value20	is-a	value
431	value20	is	arytmetic
432	calculationData4	has-a	calculationData4F2
433	calculationData4F2	is-a	calculationDataField
434	calculationData4F2	has-a	name6
435	calculationData4F2	has-a	value6
436	calculationData4	has-a	calculationData4F3
437	calculationData4F3	is-a	calculationDataField
438	calculationData4F3	has-a	name7
439	calculationData4F3	has-a	value21
440	value21	is-a	value
441	value21	is	teporaryData1
442	calculationData4	has-a	calculationData4F4
443	calculationData4F4	is-a	calculationDataField
444	calculationData4F4	has-a	name8

Line no.	Subject	Predicate	Object
445	calculationData4F4	has-a	value22
446	value22	is-a	value
447	value22	is	dimDelta
448	calculationData4	has-a	calculationData4F5
449	calculationData4F5	is-a	calculationDataField
450	calculationData4F5	has-a	name9
451	calculationData4F5	has-a	value23
452	value23	is-a	value
453	value23	is	*
454	calculationData4	has-a	calculationData4F6
455	calculationData4F6	is-a	calculationDataField
456	calculationData4F6	has-a	name10
457	calculationData4F6	has-a	value10
458	Recipe3S6	has-a	Recipe3S7
459	Recipe3S7	is-a	step
460	Recipe3S7	has-a	calculateData5
461	calculateData5	is-a	action
462	calculateData5	has-a	calculate
463	calculateData5	has-a	calculationData5
464	calculationData5	is-a	calculationData
465	calculationData5	has-a	calculationData5F1
466	calculationData5F1	is-a	calculationDataField
467	calculationData5F1	has-a	name5
468	calculationData5F1	has-a	value24
469	value24	is-a	value
470	value24	is	arytmetic
471	calculationData5	has-a	calculationData5F2
472	calculationData5F2	is-a	calculationDataField
473	calculationData5F2	has-a	name6
474	calculationData5F2	has-a	value6
475	calculationData5	has-a	calculationData5F3
476	calculationData5F3	is-a	calculationDataField
477	calculationData5F3	has-a	name7
478	calculationData5F3	has-a	value25
479	value25	is-a	value
480	value25	is	teporaryData1
481	calculationData5	has-a	calculationData5F4
482	calculationData5F4	is-a	calculationDataField
483	calculationData5F4	has-a	name8
484	calculationData5F4	has-a	value26
485	value26	is-a	value
486	value26	is	100
487	calculationData5	has-a	calculationData5F5
488	calculationData5F5	is-a	calculationDataField
489	calculationData5F5	has-a	name9
490	calculationData5F5	has-a	value27
491	value27	is-a	value
492	value27	is	/
493	calculationData5	has-a	calculationData5F6
494	calculationData5F6	is-a	calculationDataField

Line no.	Subject	Predicate	Object
495	calculationData5F6	has-a	name10
496	calculationData5F6	has-a	value10
497	Recipe3S7	has-a	Recipe3S11
498	Recipe3S8	is-a	step
499	Recipe3S8	has-a	calculateData6
500	calculateData6	is-a	action
501	calculateData6	has-a	calculate
502	calculateData6	has-a	calculationData6
503	calculationData6	is-a	calculationData
504	calculationData6	has-a	calculationData6F1
505	calculationData6F1	is-a	calculationDataField
506	calculationData6F1	has-a	name5
507	calculationData6F1	has-a	value28
508	value28	is-a	value
509	value28	is	arytmetic
510	calculationData6	has-a	calculationData6F2
511	calculationData6F2	is-a	calculationDataField
512	calculationData6F2	has-a	name6
513	calculationData6F2	has-a	value6
514	calculationData6	has-a	calculationData6F3
515	calculationData6F3	is-a	calculationDataField
516	calculationData6F3	has-a	name7
517	calculationData6F3	has-a	value29
518	value29	is-a	value
519	value29	is	teporaryData1
520	calculationData6	has-a	calculationData6F4
521	calculationData6F4	is-a	calculationDataField
522	calculationData6F4	has-a	name8
523	calculationData6F4	has-a	value30
524	value30	is-a	value
525	value30	is	dimValue
526	calculationData6	has-a	calculationData6F5
527	calculationData6F5	is-a	calculationDataField
528	calculationData6F5	has-a	name9
529	calculationData6F5	has-a	value31
530	value31	is-a	value
531	value31	is	+
532	calculationData6	has-a	calculationData6F6
533	calculationData6F6	is-a	calculationDataField
534	calculationData6F6	has-a	name10
535	calculationData6F6	has-a	value10
536	Recipe3S8	has-a	Recipe3S9
537	Recipe3S10	has-a	calculateData7
538	calculateData7	is-a	action
539	calculateData7	has-a	calculate
540	calculateData7	has-a	calculationData7
541	calculationData7	is-a	calculationData
542	calculationData7	has-a	calculationData7F1
543	calculationData7F1	is-a	calculationDataField
544	calculationData7F1	has-a	name5

Line no.	Subject	Predicate	Object
545	calculationData7F1	has-a	value34
546	value34	is-a	value
547	value34	is	function
548	calculationData7	has-a	calculationData7F3
549	calculationData7F3	is-a	calculationDataField
550	calculationData7F3	has-a	name7
551	calculationData7F3	has-a	value35
552	value35	is-a	value
553	value35	is	functionStepData
554	calculationData7	has-a	calculationData7F5
555	calculationData7F5	is-a	calculationDataField
556	calculationData7F5	has-a	name9
557	calculationData7F5	has-a	value36
558	value36	is-a	value
559	value36	is	function2
560	calculationData7	has-a	calculationData7F6
561	calculationData7F6	is-a	calculationDataField
562	calculationData7F6	has-a	name10
563	calculationData7F6	has-a	value10
564	Recipe3S10	has-a	Recipe3S6
565	Recipe3S11	is-a	step
566	Recipe3S11	has-a	skipSteps2
567	skipSteps2	is-a	action
568	skipSteps2	has-a	skip
569	skipSteps2	has-a	teporaryData2
570	skipSteps2	has-a	SkipData2
571	SkipData2	is-a	SkipData
572	SkipData2	has-a	SkipDataField3
573	SkipDataField3	is-a	SkipDataField
574	SkipDataField3	has-a	name11
575	SkipDataField3	has-a	value32
576	value32	is-a	value
577	value32	is	Recipe3S8
578	SkipData2	has-a	SkipDataField4
579	SkipDataField4	is-a	SkipDataField
580	SkipDataField4	has-a	name12
581	SkipDataField4	has-a	value33
582	value33	is-a	value
583	value33	is	Recipe3S12
584	Recipe3S11	has-a	Recipe3S8
585	Recipe3S12	is-a	step
586	Recipe3S12	has-a	calculateData8
587	calculateData8	is-a	action
588	calculateData8	has-a	calculate
589	calculateData8	has-a	calculationData8
590	calculationData8	is-a	calculationData
591	calculationData8	has-a	calculationData8F1
592	calculationData8F1	is-a	calculationDataField
593	calculationData8F1	has-a	name5
594	calculationData8F1	has-a	value37

Line no.	Subject	Predicate	Object
595	value37	is-a	value
596	value37	is	arytmetic
597	calculationData8	has-a	calculationData8F2
598	calculationData8F2	is-a	calculationDataField
599	calculationData8F2	has-a	name6
600	calculationData8F2	has-a	value6
601	calculationData8	has-a	calculationData8F3
602	calculationData8F3	is-a	calculationDataField
603	calculationData8F3	has-a	name7
604	calculationData8F3	has-a	value38
605	value38	is-a	value
606	value38	is	dimValue
607	calculationData8	has-a	calculationData8F4
608	calculationData8F4	is-a	calculationDataField
609	calculationData8F4	has-a	name8
610	calculationData8F4	has-a	value39
611	value39	is-a	value
612	value39	is	teporaryData1
613	calculationData8	has-a	calculationData8F5
614	calculationData8F5	is-a	calculationDataField
615	calculationData8F5	has-a	name9
616	calculationData8F5	has-a	value40
617	value40	is-a	value
618	value40	is	-
619	calculationData8	has-a	calculationData8F6
620	calculationData8F6	is-a	calculationDataField
621	calculationData8F6	has-a	name10
622	calculationData8F6	has-a	value10
623	Recipe3S12	has-a	Recipe3S9
624	Recipe3S13	is-a	step
625	Recipe3S13	has-a	calculateData9
626	calculateData9	is-a	action
627	calculateData9	has-a	calculate
628	calculateData9	has-a	calculationData9
629	calculationData9	is-a	calculationData
630	calculationData9	has-a	calculationData9F1
631	calculationData9F1	is-a	calculationDataField
632	calculationData9F1	has-a	name5
633	calculationData9F1	has-a	value41
634	value41	is-a	value
635	value41	is	arytmetic
636	calculationData9	has-a	calculationData9F2
637	calculationData9F2	is-a	calculationDataField
638	calculationData9F2	has-a	name6
639	calculationData9F2	has-a	value6
640	calculationData9	has-a	calculationData9F3
641	calculationData9F3	is-a	calculationDataField
642	calculationData9F3	has-a	name7
643	calculationData9F3	has-a	value42
644	value42	is-a	value

Line no.	Subject	Predicate	Object
645	value42	is	idealIntensity
646	calculationData9	has-a	calculationData9F4
647	calculationData9F4	is-a	calculationDataField
648	calculationData9F4	has-a	name8
649	calculationData9F4	has-a	value43
650	value43	is-a	value
651	value43	is	0
652	calculationData9	has-a	calculationData9F5
653	calculationData9F5	is-a	calculationDataField
654	calculationData9F5	has-a	name9
655	calculationData9F5	has-a	value44
656	value44	is-a	value
657	value44	is	+
658	calculationData9	has-a	calculationData9F6
659	calculationData9F6	is-a	calculationDataField
660	calculationData9F6	has-a	name10
661	calculationData9F6	has-a	value10
662	Recipe3S13	has-a	Recipe3S14
663	Recipe3S14	is-a	step
664	Recipe3S14	has-a	calculateData10
665	calculateData10	is-a	action
666	calculateData10	has-a	calculate
667	calculateData10	has-a	calculationData10
668	calculationData10	is-a	calculationData
669	calculationData10	has-a	calculationData10F1
670	calculationData10F1	is-a	calculationDataField
671	calculationData10F1	has-a	name5
672	calculationData10F1	has-a	value45
673	value45	is-a	value
674	value45	is	compare
675	calculationData10	has-a	calculationData10F2
676	calculationData10F2	is-a	calculationDataField
677	calculationData10F2	has-a	name6
678	calculationData10F2	has-a	value6
679	calculationData10	has-a	calculationData10F3
680	calculationData10F3	is-a	calculationDataField
681	calculationData10F3	has-a	name7
682	calculationData10F3	has-a	value46
683	value46	is-a	value
684	value46	is	lightIntensityValue
685	calculationData10	has-a	calculationData10F4
686	calculationData10F4	is-a	calculationDataField
687	calculationData10F4	has-a	name8
688	calculationData10F4	has-a	value47
689	value47	is-a	value
690	value47	is	teporaryData1
691	calculationData10	has-a	calculationData10F5
692	calculationData10F5	is-a	calculationDataField
693	calculationData10F5	has-a	name9
694	calculationData10F5	has-a	value48

Line no.	Subject	Predicate	Object
695	value48	is-a	value
696	value48	is	¿
697	calculationData10	has-a	calculationData10F6
698	calculationData10F6	is-a	calculationDataField
699	calculationData10F6	has-a	name10
700	calculationData10F6	has-a	value10
701	Recipe3S14	has-a	Recipe3S15
702	Recipe3S15	is-a	step
703	Recipe3S15	has-a	skipSteps3
704	skipSteps3	is-a	action
705	skipSteps3	has-a	skip
706	skipSteps3	has-a	teporaryData1
707	skipSteps3	has-a	SkipData3
708	SkipData3	is-a	SkipData
709	SkipData3	has-a	SkipDataField5
710	SkipDataField5	is-a	SkipDataField
711	SkipDataField5	has-a	name11
712	SkipDataField5	has-a	value49
713	value49	is-a	value
714	value49	is	Recipe3S10
715	SkipData3	has-a	SkipDataField6
716	SkipDataField6	is-a	SkipDataField
717	SkipDataField6	has-a	name12
718	SkipDataField6	has-a	value50
719	value50	is-a	value
720	value50	is	stepStop
721	Recipe3S15	has-a	Recipe3S10
722	Recipe3S9	is-a	step
723	Recipe3S9	has-a	SetDevice1
724	SetDevice1	is-a	action
725	SetDevice1	has-a	deviceFunction
726	SetDevice1	has-a	dimX
727	dimX	is-a	dim
728	dimX	is	teporaryData1
729	Recipe3S9	has-a	stepStop
730	RecipeR4	is-a	recipe
731	RecipeR4	has-a	recipeType5
732	recipeType5	is-a	recipeType
733	recipeType5	is	getUpgrated
734	RecipeR4	has-a	RecipeR4S1
735	RecipeR4S1	is-a	step
736	RecipeR4S1	has-a	RespondUpgradeRequestR4
737	RespondUpgradeRequestR4	is-a	action
738	RespondUpgradeRequestR4	has-a	massageOut
739	RespondUpgradeRequestR4	has-a	messageR41
740	messageR41	is-a	ackMessage
741	messageR41	has-a	fieldR41
742	fieldR41	is-a	messageField
743	fieldR41	has-a	idR41
744	idR41	is-a	id

Line no.	Subject	Predicate	Object
745	idR41	is	5
746	fieldR41	has-a	valueR41
747	valueR41	is-a	value
748	valueR41	is	0
749	fieldR41	has-a	nameR41
750	nameR41	is-a	name
751	nameR41	is	ackResponse
752	RecipeR4S1	has-a	RecipeR4S2
753	RecipeR4S2	is-a	step
754	RecipeR4S2	has-a	ReceiveTransmissionR4
755	ReceiveTransmissionR4	is-a	action
756	ReceiveTransmissionR4	has-a	receiveTransmission
757	RecipeR4S2	has-a	RecipeR4S3
758	RecipeR4S3	is-a	step
759	RecipeR4S3	has-a	RespondTransmissionR4
760	RespondTransmissionR4	is-a	action
761	RespondTransmissionR4	has-a	massageOut
762	RespondTransmissionR4	has-a	messageR42
763	messageR42	is-a	ackMessage
764	messageR42	has-a	fieldR42
765	fieldR42	is-a	messageField
766	fieldR42	has-a	idR41
767	fieldR42	has-a	valueR42
768	valueR42	is-a	value
769	valueR42	is	0
770	fieldR42	has-a	nameR41
771	RecipeR4S3	has-a	stepStop
772	Recipe2	is-a	recipe
773	Recipe2	has-a	service3
774	service3	is-a	service
775	service3	is	lightIntensity
776	Recipe2	has-a	context3
777	context3	is-a	context
778	context3	is	indoorSensing
779	Recipe2	has-a	recipeType2
780	recipeType2	is-a	recipeType
781	recipeType2	is	startVD
782	Recipe2	has-a	person1
783	Recipe2	has-a	recipe2S1
784	recipe2S1	is-a	step
785	recipe2S1	has-a	sendRequest
786	sendRequest	is-a	action
787	sendRequest	has-a	massageOut
788	sendRequest	has-a	message5
789	message5	is-a	searchMessage
790	message5	has-a	fieldR25
791	fieldR25	is-a	messageField
792	fieldR25	has	idR25
793	idR25	is-a	id
794	idR25	is	4

Line no.	Subject	Predicate	Object
795	fieldR25	has-a	valueR25
796	valueR25	is-a	value
797	valueR25	is	configureVD
798	fieldR25	has-a	nameR25
799	nameR25	is-a	name
800	nameR25	is	RequestType
801	recipe2S1	has-a	recipe2S2
802	recipe2S2	is-a	step
803	recipe2S2	has-a	receiveVDParticipants
804	receiveVDParticipants	has-a	messageIn
805	receiveVDParticipants	has-a	participant
806	receiveVDParticipants	is-a	action
807	participant	is-a	messageAggregate
808	receiveVDParticipants	has-a	messageA2
809	messageA2	is-a	ackMessage
810	timer2	is-a	timer
811	timer2	is	1000
812	receiveVDParticipants	has-a	timer2
813	recipe2S2	has-a	recipe2S3
814	recipe2S3	is-a	step
815	recipe2S3	has-a	decideAboutConnection
816	decideAboutConnection	is-a	action
817	decideAboutConnection	has-a	decide
818	decideAboutConnection	has-a	participant
819	decideAboutConnection	has-a	rule1
820	rule1	is-a	rule
821	rule1	is	ackResponse == 1
822	recipe2S3	has-a	recipe2S4
823	recipe2S4	is-a	step
824	recipe2S4	has-a	sendAcceptance
825	sendAcceptance	is-a	action
826	sendAcceptance	has-a	massageOut
827	sendAcceptance	has-a	participant
828	sendAcceptance	has-a	message6
829	message6	is-a	acceptMessage
830	Recipe2S4	has-a	Recipe2S5
831	Recipe2S5	is-a	step
832	Recipe2S5	has-a	receiveAck
833	receiveAck	has-a	messageIn
834	receiveAck	has-a	participant
835	receiveAck	is-a	action
836	receiveAck	has-a	messageA2
837	timer3	is-a	timer
838	timer3	is	2000
839	receiveAck	has-a	timer3
840	recipe2S5	has-a	recipe2S6
841	recipe2S6	is-a	step
842	recipe2S6	has-a	finishConfiguration
843	finishConfiguration	is-a	action
844	finishConfiguration	has-a	decide

Line no.	Subject	Predicate	Object
845	finishConfiguration	has-a	participant
846	recipe2S6	has-a	stepStop
847	Recipe2S7	is-a	step
848	Recipe2S7	has-a	ActivateRecipe
849	ActivateRecipe	is-a	action
850	ActivateRecipe	has-a	poke
851	recipe2S7	has-a	stepStop
852	smart-object	has-a	capability
853	smart-object	has-a	recipe
854	smart-object	has-a	id
855	smart-object	has-a	location
856	wait	is-a	capability
857	poke	is-a	capability
858	messageIn	is-a	capability
859	massageOut	is-a	capability
860	deviceFunction	is-a	capability
861	decide	is-a	capability
862	receiveTransmission	is-a	capability
863	calculate	is-a	capability
864	skip	is-a	capability
865	saveConfigurationData	is-a	capability
866	saveGlobalData	is-a	capability
867	ConfigureRegularTask	is-a	capability
868	CheckService	is-a	capability
869	capability	has-a	command
870	message	has-a	recipeType
871	recipe	has-a	context
872	recipe	has-a	person
873	recipe	has-a	service
874	recipe	has-a	recipeType
875	recipe	has-a	step
876	step	has-a	step
877	stepStop	is-a	step
878	step	has-a	action
879	action	is-a	event
880	action	has-a	capability
881	action	has-a	message
882	action	has-a	timer
883	action	has-a	SmartData
884	action	has-a	rule
885	action	hs-a	service
886	smartDataField	has-a	name
887	smartDataField	has-a	id
888	smartDataField	has-a	value
889	messageAggregate	is-a	smartData
890	messageAggregate	has-a	message
891	configuration	has-a	messageAggregate
892	function	has-a	point
893	point	has-a	x
894	point	has-a	y

Line no.	Subject	Predicate	Object
895	calculationData	is-a	SmartData
896	calculationData	has-a	calculationDataField
897	samartData	has-a	field
898	calculationDataField	has-a	name
899	calculationDataField	has-a	value
900	calculationDataField	is-a	field
901	SkipData	is-a	SmartData
902	SkipData	has-a	SkipDataField
903	SkipDataField	has-a	name
904	SkipDataField	has-a	value
905	SimpleData	is-a	SmartData
906	SimpleDataField	has-a	name
907	SimpleDataField	has-a	value
908	SimpleDataField	is-a	field
909	value	is-a	smartData
910	message	is-a	smartData
911	message	has-a	messageField
912	messageField	is-a	smartDataField
913	messageField	has-a	id
914	messageField	has-a	name
915	messageField	has-a	value
916	message	has-a	messageType
917	searchMessage	is-a	message
918	shoutMessage	is-a	message
919	controlMessage	is-a	message
920	acceptMessage	is-a	message
921	destroyMessage	is-a	message
922	dataMessage	is-a	message
923	ackMessage	is-a	message
924	internalMessage	is-a	message
925	searchMessage	has-a	messageType0
926	messageType0	is-a	messageType
927	messageType0	is	0
928	searchMessage	has-a	HeaderF0
929	HeaderF0	is-a	messageField
930	HeaderF0	has-a	idF0
931	idF0	is-a	id
932	idF0	is	0
933	HeaderF0	has-a	nameF0
934	nameF0	is-a	name
935	nameF0	is	MessageType
936	searchMessage	has-a	HeaderF1
937	HeaderF1	is-a	messageField
938	HeaderF1	has-a	idF1
939	idF1	is-a	id
940	idF1	is	1
941	HeaderF1	has-a	nameF1
942	nameF1	is-a	name
943	nameF1	is	SenderId
944	searchMessage	has-a	HeaderF2

Line no.	Subject	Predicate	Object
945	HeaderF2	is-a	messageField
946	HeaderF2	has-a	idF2
947	idF2	is-a	id
948	idF2	is	2
949	HeaderF2	has-a	nameF2
950	nameF2	is-a	name
951	nameF2	is	VirtualDeviceId
952	searchMessage	has-a	HeaderF3
953	HeaderF3	is-a	messageField
954	HeaderF3	has-a	idF3
955	idF3	is-a	id
956	idF3	is	3
957	HeaderF3	has-a	nameF3
958	nameF3	is-a	name
959	nameF3	is	MessageId
960	searchMessage	has-a	HeaderF4
961	HeaderF4	is-a	messageField
962	HeaderF4	has-a	idF4
963	idF4	is-a	id
964	idF4	is	4
965	HeaderF4	has-a	nameF4
966	nameF4	is-a	name
967	nameF4	is	RequestType
968	searchMessage	has-a	HeaderF5
969	HeaderF5	is-a	messageField
970	HeaderF5	has-a	idF5
971	idF5	is-a	id
972	idF5	is	5
973	HeaderF5	has-a	nameF5
974	nameF5	is-a	name
975	nameF5	is	Service
976	searchMessage	has-a	HeaderF6
977	HeaderF6	is-a	messageField
978	HeaderF6	has-a	idF6
979	idF6	is-a	id
980	idF6	is	6
981	HeaderF6	has-a	nameF6
982	nameF6	is-a	name
983	nameF6	is	Context
984	searchMessage	has-a	HeaderF7
985	HeaderF7	is-a	messageField
986	HeaderF7	has-a	idF7
987	idF7	is-a	id
988	idF7	is	7
989	HeaderF7	has-a	nameF7
990	nameF7	is-a	name
991	nameF7	is	Person
992	searchMessage	has-a	HeaderF8
993	HeaderF8	is-a	messageField
994	HeaderF8	has-a	idF8

Line no.	Subject	Predicate	Object
995	idF8	is-a	id
996	idF8	is	8
997	HeaderF8	has-a	nameF8
998	nameF8	is-a	name
999	nameF8	is	Priority
1000	acceptMessage	has-a	messageType1
1001	messageType1	is-a	messageType
1002	messageType1	is	1
1003	acceptMessage	has-a	HeaderF0
1004	acceptMessage	has-a	HeaderF1
1005	acceptMessage	has-a	HeaderF2
1006	acceptMessage	has-a	HeaderF3
1007	acceptMessage	has-a	HeaderF9
1008	HeaderF9	is-a	messageField
1009	HeaderF9	has-a	idF4
1010	HeaderF9	has-a	nameF9
1011	nameF9	is-a	name
1012	nameF9	is	RecipientId
1013	ackMessage	has-a	messageType2
1014	messageType2	is-a	messageType
1015	messageType2	is	2
1016	ackMessage	has-a	HeaderF0
1017	ackMessage	has-a	HeaderF1
1018	ackMessage	has-a	HeaderF2
1019	ackMessage	has-a	HeaderF3
1020	ackMessage	has-a	HeaderF10
1021	HeaderF10	is-a	messageField
1022	HeaderF10	has-a	idF4
1023	HeaderF10	has-a	nameF10
1024	nameF10	is-a	name
1025	nameF10	is	OryginalMessageId
1026	ackMessage	has-a	HeaderF11
1027	HeaderF11	is-a	messageField
1028	HeaderF11	has-a	idF5
1029	HeaderF11	has-a	nameF11
1030	nameF11	is-a	name
1031	nameF11	is	AckResponse
1032	shoutMessage	has-a	messageType3
1033	messageType3	is-a	messageType
1034	messageType3	is	3
1035	shoutMessage	has-a	HeaderF0
1036	shoutMessage	has-a	HeaderF1
1037	shoutMessage	has-a	HeaderF2
1038	shoutMessage	has-a	HeaderF3
1039	shoutMessage	has-a	HeaderF4
1040	shoutMessage	has-a	HeaderF5
1041	shoutMessage	has-a	HeaderF6
1042	shoutMessage	has-a	HeaderF7
1043	shoutMessage	has-a	HeaderF8
1044	destroyMessage	has-a	messageType4

Line no.	Subject	Predicate	Object
1045	messageType4	is-a	messageType
1046	messageType4	is	4
1047	destroyMessage	has-a	HeaderF0
1048	destroyMessage	has-a	HeaderF1
1049	destroyMessage	has-a	HeaderF2
1050	destroyMessage	has-a	HeaderF3
1051	dataMessage	has-a	messageType6
1052	messageType6	is-a	messageType
1053	messageType6	is	6
1054	dataMessage	has-a	HeaderF0
1055	dataMessage	has-a	HeaderF1
1056	dataMessage	has-a	HeaderF2
1057	dataMessage	has-a	HeaderF3
1058	dataMessage	has-a	HeaderF12
1059	HeaderF12	is-a	messageField
1060	HeaderF12	has-a	idF4
1061	HeaderF12	has-a	nameF12
1062	nameF12	is-a	name
1063	nameF12	is	DataSize
1064	dataMessage	has-a	HeaderF13
1065	HeaderF13	is-a	messageField
1066	HeaderF13	has-a	idF5
1067	HeaderF13	has-a	nameF13
1068	nameF13	is-a	name
1069	nameF13	is	DataType
1070	dataMessage	has-a	HeaderF14
1071	HeaderF14	is-a	messageField
1072	HeaderF14	has-a	idF6
1073	HeaderF14	has-a	nameF14
1074	nameF14	is-a	name
1075	nameF14	is	DataValue
1076	controlMessage	has-a	messageType5
1077	messageType5	is-a	messageType
1078	messageType5	is	5
1079	controlMessage	has-a	HeaderF0
1080	controlMessage	has-a	HeaderF1
1081	controlMessage	has-a	HeaderF2
1082	controlMessage	has-a	HeaderF3
1083	controlMessage	has-a	HeaderF12
1084	controlMessage	has-a	HeaderF15
1085	HeaderF15	is-a	messageField
1086	HeaderF15	has-a	idF5
1087	HeaderF15	has-a	nameF15
1088	nameF15	is-a	name
1089	nameF15	is	ControlType
1090	controlMessage	has-a	HeaderF16
1091	HeaderF16	is-a	messageField
1092	HeaderF16	has-a	idF6
1093	HeaderF16	has-a	nameF16
1094	nameF16	is-a	name

Line no.	Subject	Predicate	Object
1095	nameF16	is	ControlSize
1096	controlMessage	has-a	HeaderF17
1097	HeaderF17	is-a	messageField
1098	HeaderF17	has-a	idF7
1099	HeaderF17	has-a	nameF15
1100	controlMessage	has-a	HeaderF18
1101	HeaderF18	is-a	messageField
1102	HeaderF18	has-a	idF8
1103	HeaderF18	has-a	nameF16
1104	controlMessage	has-a	HeaderF19
1105	HeaderF19	is-a	messageField
1106	HeaderF19	has-a	idF9
1107	idF9	is-a	id
1108	idF9	is	9
1109	HeaderF19	has-a	nameF15
1110	controlMessage	has-a	HeaderF20
1111	HeaderF20	is-a	messageField
1112	HeaderF20	has-a	idF10
1113	idF10	is-a	id
1114	idF10	is	10
1115	HeaderF20	has-a	nameF16
1116	internalMessage	has-a	messageType7
1117	messageType7	is-a	messageType
1118	messageType7	is	7
1119	internalMessage	has-a	HeaderF0
1120	internalMessage	has-a	HeaderF1
1121	internalMessage	has-a	HeaderF2
1122	internalMessage	has-a	HeaderF3
1123	internalMessage	has-a	HeaderF4
1124	internalMessage	has-a	HeaderF5
1125	internalMessage	has-a	HeaderF6
1126	internalMessage	has-a	HeaderF7
1127	internalMessage	has-a	HeaderF8
1128	upgrateMessage	has-a	messageType8
1129	messageType8	is-a	messageType
1130	messageType8	is	8
1131	upgrateMessage	has-a	HeaderF0
1132	upgrateMessage	has-a	HeaderF1
1133	upgrateMessage	has-a	HeaderF2
1134	upgrateMessage	has-a	HeaderF3
1135	upgrateMessage	has-a	HeaderF21
1136	HeaderF21	is-a	messageField
1137	HeaderF21	has-a	idF4
1138	HeaderF21	has-a	nameF6
1139	upgrateMessage	has-a	HeaderF5
1140	upgrateMessage	is-a	message
1141	transmitMessage	has-a	messageType9
1142	messageType9	is-a	messageType
1143	messageType9	is	9
1144	transmitMessage	has-a	HeaderF0

Line no.	Subject	Predicate	Object
1145	transmitMessage	has-a	HeaderF1
1146	transmitMessage	has-a	HeaderF2
1147	transmitMessage	has-a	HeaderF3
1148	transmitMessage	has-a	HeaderF22
1149	HeaderF22	is-a	messageField
1150	HeaderF22	has-a	idF4
1151	HeaderF22	has-a	nameF22
1152	nameF22	is-a	name
1153	nameF22	is	TransmissionSize
1154	transmitMessage	has-a	HeaderF23
1155	HeaderF23	is-a	messageField
1156	HeaderF23	has-a	idF5
1157	HeaderF23	has-a	nameF23
1158	nameF23	is-a	name
1159	nameF23	is	TripleNumber
1160	transmitMessage	has-a	HeaderF24
1161	HeaderF24	is-a	messageField
1162	HeaderF24	has-a	idF6
1163	HeaderF24	has-a	nameF24
1164	nameF24	is-a	name
1165	nameF24	is	Subject
1166	transmitMessage	has-a	HeaderF25
1167	HeaderF25	is-a	messageField
1168	HeaderF25	has-a	idF7
1169	HeaderF25	has-a	nameF25
1170	nameF25	is-a	name
1171	nameF25	is	Predicate
1172	transmitMessage	has-a	HeaderF26
1173	HeaderF26	is-a	messageField
1174	HeaderF26	has-a	idF8
1175	HeaderF26	has-a	nameF26
1176	nameF26	is-a	name
1177	nameF26	is	Object
1178	transmitMessage	is-a	message
1179	LS1	has-a	lightIntensity
1180	LS1	has-a	energyConsumption
1181	LS1	has-a	service
1182	p0	is-a	point
1183	p1	is-a	point
1184	p2	is-a	point
1185	p3	is-a	point
1186	p4	is-a	point
1187	p5	is-a	point
1188	p6	is-a	point
1189	p7	is-a	point
1190	p8	is-a	point
1191	p9	is-a	point
1192	p10	is-a	point
1193	p11	is-a	point
1194	p12	is-a	point

Line no.	Subject	Predicate	Object
1195	p13	is-a	point
1196	p14	is-a	point
1197	p15	is-a	point
1198	p16	is-a	point
1199	p17	is-a	point
1200	p18	is-a	point
1201	p19	is-a	point
1202	p20	is-a	point
1203	p21	is-a	point
1204	p22	is-a	point
1205	p23	is-a	point
1206	p24	is-a	point
1207	p25	is-a	point
1208	p26	is-a	point
1209	p27	is-a	point
1210	p28	is-a	point
1211	p29	is-a	point
1212	p30	is-a	point
1213	p31	is-a	point
1214	p32	is-a	point
1215	p33	is-a	point
1216	p34	is-a	point
1217	p35	is-a	point
1218	p36	is-a	point
1219	p37	is-a	point
1220	p38	is-a	point
1221	p39	is-a	point
1222	p40	is-a	point
1223	p41	is-a	point
1224	p42	is-a	point
1225	p43	is-a	point
1226	p44	is-a	point
1227	p45	is-a	point
1228	p46	is-a	point
1229	p47	is-a	point
1230	p48	is-a	point
1231	p49	is-a	point
1232	p50	is-a	point
1233	p51	is-a	point
1234	p52	is-a	point
1235	p53	is-a	point
1236	p54	is-a	point
1237	p55	is-a	point
1238	p56	is-a	point
1239	p57	is-a	point
1240	p58	is-a	point
1241	p59	is-a	point
1242	p60	is-a	point
1243	p61	is-a	point
1244	p62	is-a	point

Line no.	Subject	Predicate	Object
1245	p63	is-a	point
1246	p64	is-a	point
1247	p65	is-a	point
1248	p66	is-a	point
1249	p67	is-a	point
1250	p68	is-a	point
1251	p69	is-a	point
1252	p70	is-a	point
1253	p71	is-a	point
1254	p72	is-a	point
1255	p73	is-a	point
1256	p74	is-a	point
1257	p75	is-a	point
1258	p76	is-a	point
1259	p77	is-a	point
1260	p78	is-a	point
1261	p79	is-a	point
1262	p80	is-a	point
1263	p81	is-a	point
1264	p82	is-a	point
1265	p83	is-a	point
1266	p84	is-a	point
1267	p85	is-a	point
1268	p86	is-a	point
1269	p87	is-a	point
1270	p88	is-a	point
1271	p89	is-a	point
1272	p90	is-a	point
1273	p91	is-a	point
1274	p92	is-a	point
1275	p93	is-a	point
1276	p94	is-a	point
1277	p95	is-a	point
1278	p96	is-a	point
1279	p97	is-a	point
1280	p98	is-a	point
1281	p99	is-a	point
1282	p100	is-a	point
1283	p101	is-a	point
1284	p0	has-a	x0
1285	p1	has-a	x1
1286	p2	has-a	x2
1287	p3	has-a	x3
1288	p4	has-a	x4
1289	p5	has-a	x5
1290	p6	has-a	x6
1291	p7	has-a	x7
1292	p8	has-a	x8
1293	p9	has-a	x9
1294	p10	has-a	x10

Line no.	Subject	Predicate	Object
1295	p11	has-a	x11
1296	p12	has-a	x12
1297	p13	has-a	x13
1298	p14	has-a	x14
1299	p15	has-a	x15
1300	p16	has-a	x16
1301	p17	has-a	x17
1302	p18	has-a	x18
1303	p19	has-a	x19
1304	p20	has-a	x20
1305	p21	has-a	x21
1306	p22	has-a	x22
1307	p23	has-a	x23
1308	p24	has-a	x24
1309	p25	has-a	x25
1310	p26	has-a	x26
1311	p27	has-a	x27
1312	p28	has-a	x28
1313	p29	has-a	x29
1314	p30	has-a	x30
1315	p31	has-a	x31
1316	p32	has-a	x32
1317	p33	has-a	x33
1318	p34	has-a	x34
1319	p35	has-a	x35
1320	p36	has-a	x36
1321	p37	has-a	x37
1322	p38	has-a	x38
1323	p39	has-a	x39
1324	p40	has-a	x40
1325	p41	has-a	x41
1326	p42	has-a	x42
1327	p43	has-a	x43
1328	p44	has-a	x44
1329	p45	has-a	x45
1330	p46	has-a	x46
1331	p47	has-a	x47
1332	p48	has-a	x48
1333	p49	has-a	x49
1334	p50	has-a	x50
1335	p51	has-a	x0
1336	p52	has-a	x1
1337	p53	has-a	x2
1338	p54	has-a	x3
1339	p55	has-a	x4
1340	p56	has-a	x5
1341	p57	has-a	x6
1342	p58	has-a	x7
1343	p59	has-a	x8
1344	p60	has-a	x9

Line no.	Subject	Predicate	Object
1345	p61	has-a	x10
1346	p62	has-a	x11
1347	p63	has-a	x12
1348	p64	has-a	x13
1349	p65	has-a	x14
1350	p66	has-a	x15
1351	p67	has-a	x16
1352	p68	has-a	x17
1353	p69	has-a	x18
1354	p70	has-a	x19
1355	p71	has-a	x20
1356	p72	has-a	x21
1357	p73	has-a	x22
1358	p74	has-a	x23
1359	p75	has-a	x24
1360	p76	has-a	x25
1361	p77	has-a	x26
1362	p78	has-a	x27
1363	p79	has-a	x28
1364	p80	has-a	x29
1365	p81	has-a	x30
1366	p82	has-a	x31
1367	p83	has-a	x32
1368	p84	has-a	x33
1369	p85	has-a	x34
1370	p86	has-a	x35
1371	p87	has-a	x36
1372	p88	has-a	x37
1373	p89	has-a	x38
1374	p90	has-a	x39
1375	p91	has-a	x40
1376	p92	has-a	x41
1377	p93	has-a	x42
1378	p94	has-a	x43
1379	p95	has-a	x44
1380	p96	has-a	x45
1381	p97	has-a	x46
1382	p98	has-a	x47
1383	p99	has-a	x48
1384	p100	has-a	x49
1385	p101	has-a	x50
1386	p0	has-a	y0
1387	p1	has-a	y1
1388	p2	has-a	y2
1389	p3	has-a	y3
1390	p4	has-a	y4
1391	p5	has-a	y5
1392	p6	has-a	y6
1393	p7	has-a	y7
1394	p8	has-a	y8

Line no.	Subject	Predicate	Object
1395	p9	has-a	y9
1396	p10	has-a	y10
1397	p11	has-a	y11
1398	p12	has-a	y12
1399	p13	has-a	y13
1400	p14	has-a	y14
1401	p15	has-a	y15
1402	p16	has-a	y16
1403	p17	has-a	y17
1404	p18	has-a	y18
1405	p19	has-a	y19
1406	p20	has-a	y20
1407	p21	has-a	y21
1408	p22	has-a	y22
1409	p23	has-a	y23
1410	p24	has-a	y24
1411	p25	has-a	y25
1412	p26	has-a	y26
1413	p27	has-a	y27
1414	p28	has-a	y28
1415	p29	has-a	y29
1416	p30	has-a	y30
1417	p31	has-a	y31
1418	p32	has-a	y32
1419	p33	has-a	y33
1420	p34	has-a	y34
1421	p35	has-a	y35
1422	p36	has-a	y36
1423	p37	has-a	y37
1424	p38	has-a	y38
1425	p39	has-a	y39
1426	p40	has-a	y40
1427	p41	has-a	y41
1428	p42	has-a	y42
1429	p43	has-a	y43
1430	p44	has-a	y44
1431	p45	has-a	y45
1432	p46	has-a	y46
1433	p47	has-a	y47
1434	p48	has-a	y48
1435	p49	has-a	y49
1436	p50	has-a	y50
1437	p51	has-a	y51
1438	p52	has-a	y52
1439	p53	has-a	y53
1440	p54	has-a	y54
1441	p55	has-a	y55
1442	p56	has-a	y56
1443	p57	has-a	y57
1444	p58	has-a	y58

Line no.	Subject	Predicate	Object
1445	p59	has-a	y59
1446	p60	has-a	y60
1447	p61	has-a	y61
1448	p62	has-a	y62
1449	p63	has-a	y63
1450	p64	has-a	y64
1451	p65	has-a	y65
1452	p66	has-a	y66
1453	p67	has-a	y67
1454	p68	has-a	y68
1455	p69	has-a	y69
1456	p70	has-a	y70
1457	p71	has-a	y71
1458	p72	has-a	y72
1459	p73	has-a	y73
1460	p74	has-a	y74
1461	p75	has-a	y75
1462	p76	has-a	y76
1463	p77	has-a	y77
1464	p78	has-a	y78
1465	p79	has-a	y79
1466	p80	has-a	y80
1467	p81	has-a	y81
1468	p82	has-a	y82
1469	p83	has-a	y83
1470	p84	has-a	y84
1471	p85	has-a	y85
1472	p86	has-a	y86
1473	p87	has-a	y87
1474	p88	has-a	y88
1475	p89	has-a	y89
1476	p90	has-a	y90
1477	p91	has-a	y91
1478	p92	has-a	y92
1479	p93	has-a	y93
1480	p94	has-a	y94
1481	p95	has-a	y95
1482	p96	has-a	y96
1483	p97	has-a	y97
1484	p98	has-a	y98
1485	p99	has-a	y99
1486	p100	has-a	y100
1487	p101	has-a	y101
1488	x0	is-a	x
1489	x1	is-a	x
1490	x2	is-a	x
1491	x3	is-a	x
1492	x4	is-a	x
1493	x5	is-a	x
1494	x6	is-a	x

Line no.	Subject	Predicate	Object
1495	x7	is-a	x
1496	x8	is-a	x
1497	x9	is-a	x
1498	x10	is-a	x
1499	x11	is-a	x
1500	x12	is-a	x
1501	x13	is-a	x
1502	x14	is-a	x
1503	x15	is-a	x
1504	x16	is-a	x
1505	x17	is-a	x
1506	x18	is-a	x
1507	x19	is-a	x
1508	x20	is-a	x
1509	x21	is-a	x
1510	x22	is-a	x
1511	x23	is-a	x
1512	x24	is-a	x
1513	x25	is-a	x
1514	x26	is-a	x
1515	x27	is-a	x
1516	x28	is-a	x
1517	x29	is-a	x
1518	x30	is-a	x
1519	x31	is-a	x
1520	x32	is-a	x
1521	x33	is-a	x
1522	x34	is-a	x
1523	x35	is-a	x
1524	x36	is-a	x
1525	x37	is-a	x
1526	x38	is-a	x
1527	x39	is-a	x
1528	x40	is-a	x
1529	x41	is-a	x
1530	x42	is-a	x
1531	x43	is-a	x
1532	x44	is-a	x
1533	x45	is-a	x
1534	x46	is-a	x
1535	x47	is-a	x
1536	x48	is-a	x
1537	x49	is-a	x
1538	x50	is-a	x
1539	y0	is-a	y
1540	y1	is-a	y
1541	y2	is-a	y
1542	y3	is-a	y
1543	y4	is-a	y
1544	y5	is-a	y

Line no.	Subject	Predicate	Object
1545	y6	is-a	y
1546	y7	is-a	y
1547	y8	is-a	y
1548	y9	is-a	y
1549	y10	is-a	y
1550	y11	is-a	y
1551	y12	is-a	y
1552	y13	is-a	y
1553	y14	is-a	y
1554	y15	is-a	y
1555	y16	is-a	y
1556	y17	is-a	y
1557	y18	is-a	y
1558	y19	is-a	y
1559	y20	is-a	y
1560	y21	is-a	y
1561	y22	is-a	y
1562	y23	is-a	y
1563	y24	is-a	y
1564	y25	is-a	y
1565	y26	is-a	y
1566	y27	is-a	y
1567	y28	is-a	y
1568	y29	is-a	y
1569	y30	is-a	y
1570	y31	is-a	y
1571	y32	is-a	y
1572	y33	is-a	y
1573	y34	is-a	y
1574	y35	is-a	y
1575	y36	is-a	y
1576	y37	is-a	y
1577	y38	is-a	y
1578	y39	is-a	y
1579	y40	is-a	y
1580	y41	is-a	y
1581	y42	is-a	y
1582	y43	is-a	y
1583	y44	is-a	y
1584	y45	is-a	y
1585	y46	is-a	y
1586	y47	is-a	y
1587	y48	is-a	y
1588	y49	is-a	y
1589	y50	is-a	y
1590	y51	is-a	y
1591	y52	is-a	y
1592	y53	is-a	y
1593	y54	is-a	y
1594	y55	is-a	y

Line no.	Subject	Predicate	Object
1595	y56	is-a	y
1596	y57	is-a	y
1597	y58	is-a	y
1598	y59	is-a	y
1599	y60	is-a	y
1600	y61	is-a	y
1601	y62	is-a	y
1602	y63	is-a	y
1603	y64	is-a	y
1604	y65	is-a	y
1605	y66	is-a	y
1606	y67	is-a	y
1607	y68	is-a	y
1608	y69	is-a	y
1609	y70	is-a	y
1610	y71	is-a	y
1611	y72	is-a	y
1612	y73	is-a	y
1613	y74	is-a	y
1614	y75	is-a	y
1615	y76	is-a	y
1616	y77	is-a	y
1617	y78	is-a	y
1618	y79	is-a	y
1619	y80	is-a	y
1620	y81	is-a	y
1621	y82	is-a	y
1622	y83	is-a	y
1623	y84	is-a	y
1624	y85	is-a	y
1625	y86	is-a	y
1626	y87	is-a	y
1627	y88	is-a	y
1628	y89	is-a	y
1629	y90	is-a	y
1630	y91	is-a	y
1631	y92	is-a	y
1632	y93	is-a	y
1633	y94	is-a	y
1634	y95	is-a	y
1635	y96	is-a	y
1636	y97	is-a	y
1637	y98	is-a	y
1638	y99	is-a	y
1639	y100	is-a	y
1640	y101	is-a	y
1641	x0	is	0
1642	x1	is	1
1643	x2	is	2
1644	x3	is	3

Line no.	Subject	Predicate	Object
1645	x4	is	4
1646	x5	is	5
1647	x6	is	6
1648	x7	is	7
1649	x8	is	8
1650	x9	is	9
1651	x10	is	10
1652	x11	is	11
1653	x12	is	12
1654	x13	is	13
1655	x14	is	14
1656	x15	is	15
1657	x16	is	16
1658	x17	is	17
1659	x18	is	18
1660	x19	is	19
1661	x20	is	20
1662	x21	is	21
1663	x22	is	22
1664	x23	is	23
1665	x24	is	24
1666	x25	is	25
1667	x26	is	26
1668	x27	is	27
1669	x28	is	28
1670	x29	is	29
1671	x30	is	30
1672	x31	is	31
1673	x32	is	32
1674	x33	is	33
1675	x34	is	34
1676	x35	is	35
1677	x36	is	36
1678	x37	is	37
1679	x38	is	38
1680	x39	is	39
1681	x40	is	40
1682	x41	is	41
1683	x42	is	42
1684	x43	is	43
1685	x44	is	44
1686	x45	is	45
1687	x46	is	46
1688	x47	is	47
1689	x48	is	48
1690	x49	is	49
1691	x50	is	50
1692	y0	is	0
1693	y1	is	2
1694	y2	is	4

Line no.	Subject	Predicate	Object
1695	y3	is	6
1696	y4	is	8
1697	y5	is	10
1698	y6	is	12
1699	y7	is	14
1700	y8	is	16
1701	y9	is	18
1702	y10	is	20
1703	y11	is	22
1704	y12	is	24
1705	y13	is	26
1706	y14	is	28
1707	y15	is	30
1708	y16	is	32
1709	y17	is	34
1710	y18	is	36
1711	y19	is	38
1712	y20	is	40
1713	y21	is	42
1714	y22	is	44
1715	y23	is	46
1716	y24	is	48
1717	y25	is	50
1718	y26	is	52
1719	y27	is	54
1720	y28	is	56
1721	y29	is	58
1722	y30	is	60
1723	y31	is	62
1724	y32	is	64
1725	y33	is	66
1726	y34	is	68
1727	y35	is	70
1728	y36	is	72
1729	y37	is	74
1730	y38	is	76
1731	y39	is	78
1732	y40	is	80
1733	y41	is	82
1734	y42	is	84
1735	y43	is	86
1736	y44	is	88
1737	y45	is	90
1738	y46	is	92
1739	y47	is	94
1740	y48	is	96
1741	y49	is	98
1742	y50	is	100
1743	y51	is	100
1744	y52	is	98

Line no.	Subject	Predicate	Object
1745	y53	is	96
1746	y54	is	94
1747	y55	is	92
1748	y56	is	90
1749	y57	is	88
1750	y58	is	86
1751	y59	is	84
1752	y60	is	82
1753	y61	is	80
1754	y62	is	78
1755	y63	is	76
1756	y64	is	74
1757	y65	is	72
1758	y66	is	70
1759	y67	is	68
1760	y68	is	66
1761	y69	is	64
1762	y70	is	62
1763	y71	is	60
1764	y72	is	58
1765	y73	is	56
1766	y74	is	54
1767	y75	is	52
1768	y76	is	50
1769	y77	is	48
1770	y78	is	46
1771	y79	is	44
1772	y80	is	42
1773	y81	is	40
1774	y82	is	38
1775	y83	is	36
1776	y84	is	34
1777	y85	is	32
1778	y86	is	30
1779	y87	is	28
1780	y88	is	26
1781	y89	is	24
1782	y90	is	22
1783	y91	is	20
1784	y92	is	18
1785	y93	is	16
1786	y94	is	14
1787	y95	is	12
1788	y96	is	10
1789	y97	is	8
1790	y98	is	6
1791	y99	is	4
1792	y100	is	2
1793	y101	is	0
1794	function1	is-a	function

Line no.	Subject	Predicate	Object
1795	function1	has-a	p0
1796	function1	has-a	p1
1797	function1	has-a	p2
1798	function1	has-a	p3
1799	function1	has-a	p4
1800	function1	has-a	p5
1801	function1	has-a	p6
1802	function1	has-a	p7
1803	function1	has-a	p8
1804	function1	has-a	p9
1805	function1	has-a	p10
1806	function1	has-a	p11
1807	function1	has-a	p12
1808	function1	has-a	p13
1809	function1	has-a	p14
1810	function1	has-a	p15
1811	function1	has-a	p16
1812	function1	has-a	p17
1813	function1	has-a	p18
1814	function1	has-a	p19
1815	function1	has-a	p20
1816	function1	has-a	p21
1817	function1	has-a	p22
1818	function1	has-a	p23
1819	function1	has-a	p24
1820	function1	has-a	p25
1821	function1	has-a	p26
1822	function1	has-a	p27
1823	function1	has-a	p28
1824	function1	has-a	p29
1825	function1	has-a	p30
1826	function1	has-a	p31
1827	function1	has-a	p32
1828	function1	has-a	p33
1829	function1	has-a	p34
1830	function1	has-a	p35
1831	function1	has-a	p36
1832	function1	has-a	p37
1833	function1	has-a	p38
1834	function1	has-a	p39
1835	function1	has-a	p40
1836	function1	has-a	p41
1837	function1	has-a	p42
1838	function1	has-a	p43
1839	function1	has-a	p44
1840	function1	has-a	p45
1841	function1	has-a	p46
1842	function1	has-a	p47
1843	function1	has-a	p48
1844	function1	has-a	p49

Line no.	Subject	Predicate	Object
1845	function1	has-a	p50
1846	function2	is-a	function
1847	function2	has-a	p51
1848	function2	has-a	p52
1849	function2	has-a	p53
1850	function2	has-a	p54
1851	function2	has-a	p55
1852	function2	has-a	p56
1853	function2	has-a	p57
1854	function2	has-a	p58
1855	function2	has-a	p59
1856	function2	has-a	p60
1857	function2	has-a	p61
1858	function2	has-a	p62
1859	function2	has-a	p63
1860	function2	has-a	p64
1861	function2	has-a	p65
1862	function2	has-a	p66
1863	function2	has-a	p67
1864	function2	has-a	p68
1865	function2	has-a	p69
1866	function2	has-a	p70
1867	function2	has-a	p71
1868	function2	has-a	p72
1869	function2	has-a	p73
1870	function2	has-a	p74
1871	function2	has-a	p75
1872	function2	has-a	p76
1873	function2	has-a	p77
1874	function2	has-a	p78
1875	function2	has-a	p79
1876	function2	has-a	p80
1877	function2	has-a	p81
1878	function2	has-a	p82
1879	function2	has-a	p83
1880	function2	has-a	p84
1881	function2	has-a	p85
1882	function2	has-a	p86
1883	function2	has-a	p87
1884	function2	has-a	p88
1885	function2	has-a	p89
1886	function2	has-a	p90
1887	function2	has-a	p91
1888	function2	has-a	p92
1889	function2	has-a	p93
1890	function2	has-a	p94
1891	function2	has-a	p95
1892	function2	has-a	p96
1893	function2	has-a	p97
1894	function2	has-a	p98

Line no.	Subject	Predicate	Object
1895	function2	has-a	p99
1896	LS1	has-a	energyConsumption
1897	LS1	has-a	service
1898	LS1	has-a	lightIntensity
1899	LS1	has-a	margin
1900	LS1	has-a	dimdelta

B Appendix B: Recipe Conditional Choices Diagrams

A diagram in Figure 54 presents a recipe from a lamp (see also Appendix A, Listing 250-729). A diagram in Figure 55 presents an example of a recipie with conditional choices in the Mirror's knowledgebase.

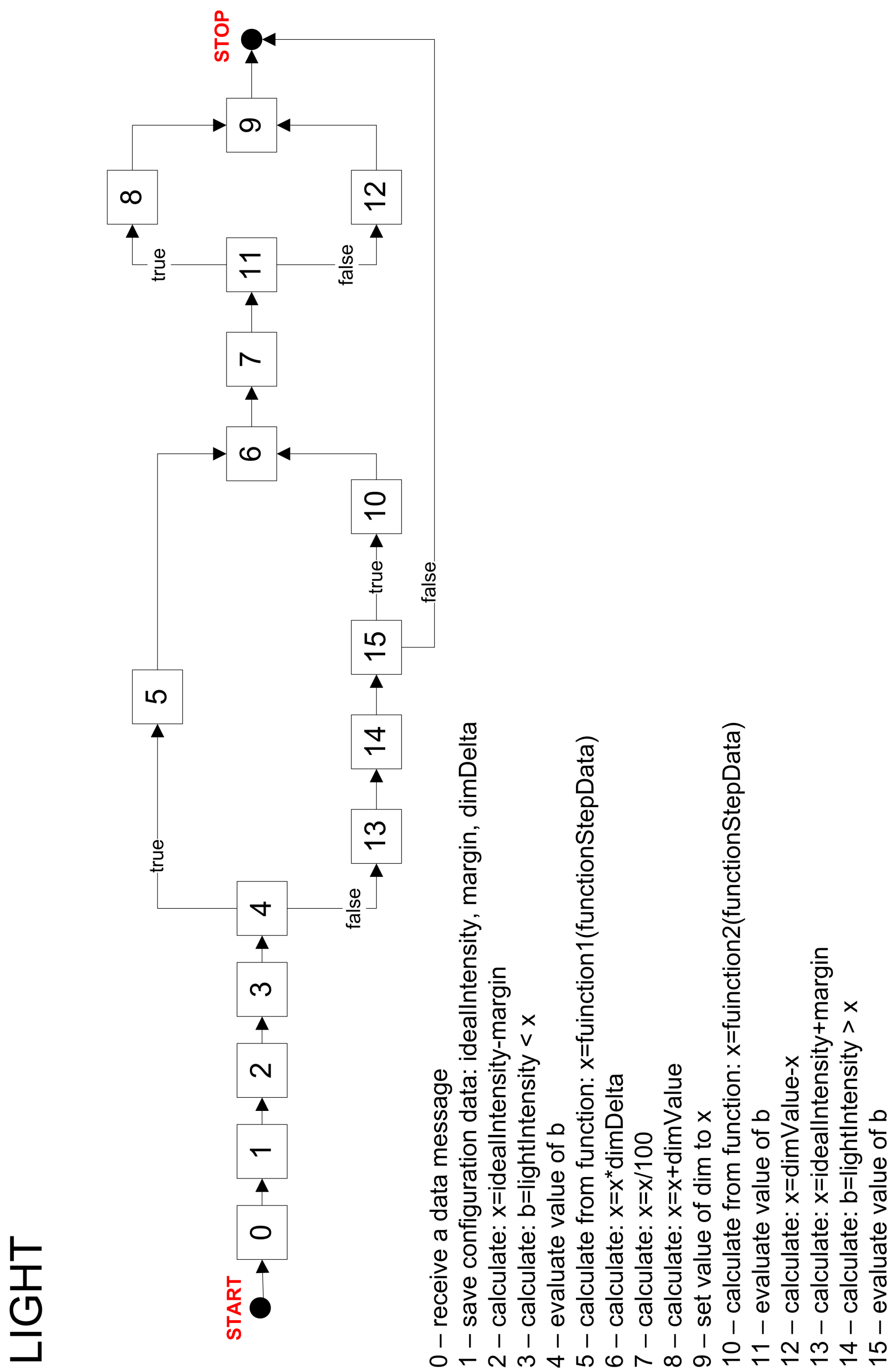
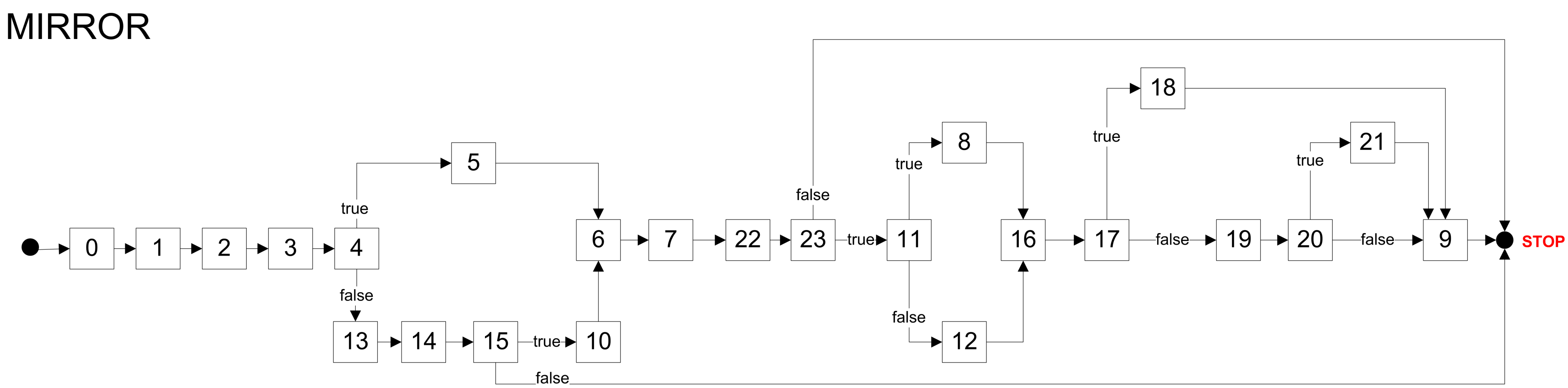


Figure 54: Lamp's *Recipe3* calculating a dim level.



0 – receive a data message
1 – save configuration data: idealIntensity, margin, dimDelta
2 – calculate: x=idealIntensity-margin
3 – calculate: b=lightIntensity < x
4 – evaluate value of b
5 – calculate from function: x=fuinction1(functionStepData)
6 – calculate: x=x*dimDelta
7 – calculate: x=x/100
8 – calculate: x=x+dimValue
9 – set value of dim to x
10 – calculate from function: x=fuinction2(functionStepData)
11 – evaluate value of b
12 – calculate: x=dimValue-x
13 – calculate: x=idealIntensity+margin
14 – calculate: b=lightIntensity > x
15 – evaluate value of b
16 – calculate: b=x>MaxIntensity
17 – evaluate value of b
18 – save data: x=MaxIntensity
19 – calculate: b=x<0
20 – evaluate value of b
21 – save data: x=0
22 – calculate: b=x>0
23 – evaluate value of b

Figure 55: Mirrors's recipe calculating a dim level.

C Appendix C: List of Publications

- **Modelling a Multi-Core Media Processor Using JCSP** by Jon M. Kerridge, Anna Kosek, and Aly Syed. Published in proceedings of Communicating Process Architectures in 2008 (CPA'08) [34].

  Abstract. Manufacturers are creating multi-core processors to solve specialized problems. This kind of processor can process tasks faster by running them in parallel. This paper explores the usability of the Communicating Sequential Processes model to create a simulation of a multi-core processor aimed at media processing in hand-held mobile devices. Every core in such systems can have different capabilities and can generate different amounts of heat depending on the task being performed. Heat generated reduces the performance of the core. We have used mobile processes in JCSP to implement the allocation of tasks to cores based upon the work the core has done previously.

- **JCSP agents-based service discovery for pervasive computing** by Jon M. Kerridge, Anna Kosek, and Aly Syed. Published in proceedings of Communicating Process Architectures in 2009 (CPA'09) [35].

  Abstract. Device and service discovery is a very important topic when considering pervasive environments. The discovery mechanism is required to work in networks with dynamic topology and on limited software, and be able to accept different device descriptions. This paper presents a service and device discovery mechanism using JCSP agents and the JCSP network package *jcsp.net2*.

- **A dynamic connection capability for pervasive adaptive environments using JCSP** by Anna Kosek, Aly Syed, Jon Kerridge, and Alistair Armitage. Published in proceedings of The Thirty Fifth Annual Convention of the Society for the Study of Artificial Intelligence and Simulation of Behaviour in 2009 (AISB'09).[37]

  Abstract. The house, office or warehouse environment is full of devices that make users' life and work easier. People nowadays use personal computers, laptops, Personal Digital Assistants, mobile phones and many more devices with ease. The mechanism to connect, enable co-operation and exchange data between devices will help to use devices' full capabilities. This paper investigates the usability of Communicating Sequential Processes for Java in pervasive systems and the adaptation possibilities offered by this environment. It focuses on dynamic connection capabilities. The paper also describes an experiment that organizes an adapting pervasive environment which uses dynamic connections for data flow.

- **RDF Recipes for Context-Aware Interoperability in Pervasive Systems** by Anna M. Kosek, Aly A. Syed, and Jon M. Kerridge. Published in proceedings of IEEE Symposium on Computers and Communications in 2010 (ISCC'10).[38]

  Abstract. Nowadays home automation systems integrate many devices from security system, heating, ventilation and air conditioning system, lighting system or audio-video systems. Every time a new device is installed, problems with connecting it to other devices and synchronization may appear. The technology trends are to build more powerful and functional new autonomous devices, rather than improve device synchronization, data and functional distribution and adaptation to changing environment. This paper highlights interoperability problems in pervasive computing and presents a solution for devices with limited processing capabilities by use of an ontology for knowledge formulation and semantic interoperability.