Engineering MAS – A Device Integration Framework for Smart Home Environmentss

Jack Betts and Berndt Müller

University of South Wales, United Kingdom
{jack.betts,bertie.muller}@southwales.ac.uk

Abstract. We introduce a layered approach to multi-agent programming and motivate this with a perspective to smart home environments. Apart from the core layer, layer components can be updated at runtime to reflect, e.g., attributes like credibility and the addition of proprietary functionality. The Layered Agent Framework (LAF) is defined by interfaces and organised into layers. This approach minimises system fragmentation while allowing developers to create and maintain meaningful and effective agents. A Petri net model is provided to visualise and execute prototypes of the agents. Although fully functional, the Petri nets will later be translated into dedicated programs with a smaller footprint and more efficient execution.

1 Introduction

The paper discusses a flexible, dynamic, and software-developer friendly framework for the development of multi-agent systems with the application domain of home automation and assistance in mind. The resulting framework will be easily adaptable for other domains, since the smart home encompasses a multitude of technologies and has many requirements - such as real-time interaction - also found in other areas.

The remainder of this section will give an introduction to the application domain and the problems we are addressing with our approach. Section 3 presents our approach to a layered agent-based framework for the integration of devices in the home-automation domain. Some notes on the implementation can be found in Section 3.3. We close with an outlook on the next steps and possible further developments in Section 4.

1.1 What is a Smart Home?

When reading about smart devices or smart solutions, usually very little is revealed about the actual methodology used to make them ‘smart’. In some cases this may be due to the fact that the developer wants to conceal the fact that they are using agent technology, in other cases the marketing of a ‘smart’ service may distract from reality and from the fact that the smartness of the service is just an added timer or a simple remote control on a mobile phone. When we think of a smart home, we use the term ‘smart’ to denote the use of intelligent solutions that would help organise our everyday life in and around the home. So, what would be involved? Rather than a timer giving the illusion of an intelligent system agent technology should be utilised to enable the ‘smart

home' to learn from the user and adapt to better help the user. This learning should be unobtrusive from the perspective of the user. User interaction will be encouraged but not enforced in order for the system to learn quicker.[1] The following is a (probably not exhaustive) list of areas affected by smart-home technology:

- energy systems and power consumption
- entertainment systems (audio, video, and text)
- health support systems
- household appliances
- shopping and banking transactions
- education and reference sourcing
- news provision

From this list it becomes clear that a wide range of areas have to be covered and many different requirements have to be addressed. Our aim is to provide an infrastructure to enable reasoning and communication amongst all entities involved in these areas. This includes stationary and mobile entities situated within the home and mobile entities that may enter and leave the home environment. A change of environments may require different representation of data or data dependencies [1] and this has to be supported by the framework.

1.2 The role of AI in a Smart Home

Focussing on smart home applications, we are faced with a variety of AI techniques that can be employed to assist the user in an unobtrusive, reliable, and helpful way. Our main focus is on agent orientation and the use of multiple (mobile) agents. To achieve the level of integration required to address all (or at least most) of the challenges, we have to allow for different types of communication and different types of actions. Communication and stored data have to be encrypted where necessary and to an appropriate standard. Access to parts of the system has to be authenticated and regimented.

1.3 Core Areas

The core areas supported by our system architecture are

- cooperation by communication
- fault tolerance
- user experience
- intelligent unobtrusiveness.

Communication is a foundation for all of these core areas. Unobtrusive assistance can only be achieved if the information to be provided is available everywhere at any time[2], so that the optimal (or near optimal) moment and place can be chosen to convey this information to the user (or agent). Fault tolerance also requires communication in order to re-schedule tasks while a device is unavailable. Methods to support the core functionality are part of a core layer in our design.

[1] In this paper we are not concerned with the actual learning or the provision of the intelligence, but rather with an architecture to support the development of such systems.

[2] This is an idealised vision. In practice it will be sufficient to have the possibility of the information being made available at many locations in the home most of the time.

1.4 What is new in our approach?

The methodology introduced in this paper is designed to help the software developer by supplying familiar concepts from the world of object orientation to that of agent orientation without making everything behave like an agent. We introduce several layers with notions of inheritance and overriding as known from established object oriented frameworks. Components and patterns are encouraged for the supply of common agent functionality. For example, agent or task generation can be accomplished by instantiation from a blueprint, much alike the creation of an object from a class.

The way the architecture is designed has dynamic systems and an excellent user experience in mind. Adding or changing functionality as necessary within the running system is provided on layers on top of the agent core. These layers can be modified dynamically without the need of restarting the system, hence integrating seamlessly in the user's workflow. The architecture can support agents can keep previous copies of protocols, roles, and layer-specific functionality in case an update fails. Hereby, an immediate rollback to a previous version of a layer is possible and a report the update as failed can be communicated without making the system inoperative. For the smart home environment, this means unobtrusive updates and structural changes are possible without noticeable interruptions of the overall system's functionality.

2 Summarising the *status quo*

We briefly summarise some existing work on smart home automation. This is not meant to be a full account of the literature, but serves as an indication of different approaches. Section 2.1 discusses AI techniques used in various flavours of home automation. Section 2.2 gives an overview of agent-based approaches. In both sections, we point out which aspects have influenced the design decisions for the model presented in Section 3.

2.1 General Approaches

In the past decade, home automation has been discussed at various levels. Some approaches involve AI techniques [2, 3], others are looking at home integration from a predominantly (electrical-)engineering perspective (e.g. [4]) or from a sociological viewpoint (e.g. [5]).

Many approaches tackle only one aspect of home automation, e.g. the heating system or power consumption.

2.2 Agent-based Approaches

A multitude of papers discuss energy management or other specific home automation systems. One of these, [6], makes the point that the home automation market is fragmented. Different technologies compete, are incompatible and co-exist, rather than cooperate. It is argued that interoperation can be achieved by means of an abstraction layer that would allow access to different home automation devices in a uniform and generic

way, independent of the underlying technology. [7] is concerned with resource allocation and optimisation using an abstract negotiation protocol. Agents negotiate near optimal settings for minimising power consumption to reduce the greenhouse effect by integrating appliances and heating systems. The paper remains at an academic level in that no solutions are offered for an implementation in a real-world situation.

Other approaches are more general and in line with the present endeavour, but remain unspecific. [8] describes a home automation system called HORUS, which is based on an agent architecture including managers, IO handles, video-camera handlers, and alarm communicators. The IP-communication-based system remains rather vague about the format of rules and their interpretation in a setting with existing appliances.

3 Our Approach

The centrepiece of our approach is a component-based and software-developer friendly system architecture that will allow legacy systems to be integrated into a multi-agent-based framework built around the requirements of smart home automation. As such it will allow the developer to dynamiacally incorporate many aspects including *learning of behavioural patterns*, *handling of sensitive data*, *unobtrusive conveyance of information and general assistance*, and *monitoring of various sensors*.

The software development approach is based on object orientated design, but does not simply replace objects with agents as some approaches in the past have[3]. Instead, we use agents alongside traditional objects to reflect different capabilities of the various system components. In doing so, we avoid having to discard legacy components and re-design them for an all-agents system. Also, we avoid unnecessary communications complexity that an all agents approach would incur.

The main reason for focusing our approach on the smart home environment is that future real-world use of agent technology will only be of benefit when technologies and requirements meet and a (more or less) seamless interaction is guaranteed. This is of foremost importance and means that isolated studies are generally not scalable. By providing the interfaces for the use of modern agent-based interaction we open up possibilities without restricting the use of more traditional software and hardware paradigms. For the home environment, this means that new devices and appliances might be constructed to include some additional inexpensive hardware/software components to allow integration with others, while legacy components may be integrated by simple plug-ons (software and hardware), such as a communications-enabled wall-plug adapter that has some basic control over an appliance (e.g., switching it on or off or monitoring power draw and detecting usage patterns of the socket and making this information accessible this information to the agent network.).

The components introduced in this paper will be part of a home system design consisting of stationary and mobile devices and learning control systems based on software agents. The stationary devices are in the main part traditional household appliances whose controls become part of a dynamically learning distributed control system including some stationary devices (perceptors, like motion and temperature sensors; control hubs, acting somewhat like servers on which computationally expensive tasks of

[3] "Agents can be seen as the successors of objects and classes ...", `http://aose.org`

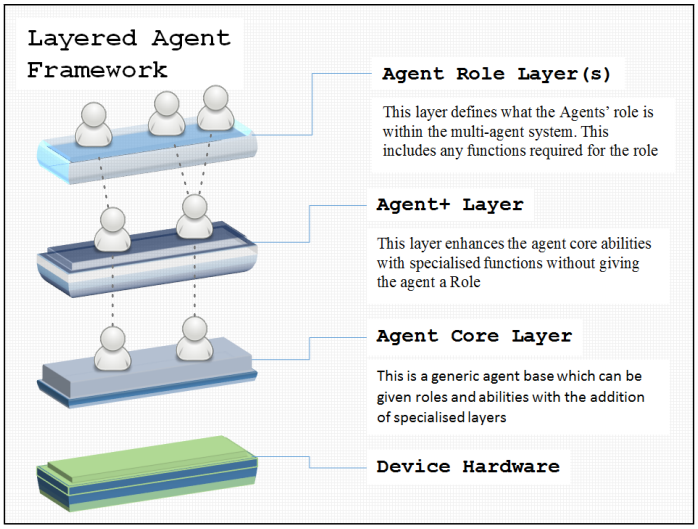

Fig. 1. Layered Agent Framework

learning and processing of data is done) and mobile devices migrating between environments and allowing the overall system to learn behavioural patterns as well as hosting agents themselves to carry out independent tasks.

3.1 Theoretical Model

When creating multi-agent systems a common platform on which the agents are built is necessary. This may be a common communication protocol to enable all the agents to talk to each other. This can be a common runtime like a Java Virtual Machine (JVM), or an operating system (Linux) providing a common platform for all agents. Previous research has brought about a common intermediate layer (MCAPL) that executes agents programmed in a variety of agent programming languages after translation [9, 10].

Common communication elements and interfaces are required to enable agents to work together, however different multi-agent systems developed by different development teams generally cannot communicate effectively with each other without specific adapters. A simple solution would be to use a common open communication protocol and declare this a standard for all agent communications[4]. This is a solution only if all agents on all systems followed this standard, and that the protocol offered everything that all MAS developers need.

This leads to the question: How is it possible to allow agent developers as much freedom as possible without imposing too many restrictions on their code? Our approach is the design of a Layered Agent Framework (LAF, see Figure 1) that provides developers with a generic agent layer that can be specialised in order to fulfil defined roles. This specialisation is achieved through the addition of further layers. The system

[4] http://www.fipa.org

is not limited to one role per agent; the *Agent Role Layers* can be stacked to provide a multi-role agent system. The ordering of the role layers determines the priorities of the agent. Any role layers that are packaged with the system are default roles. Default roles can not be dropped by the system and cannot be superseded in priority by non default role layers (default role priority set by the developer or the system). This layered approach loosely couples additional roles to an agent, allowing roles to be changed, updated or multiplied without requiring the agent to be fully restarted. Not only are the agent roles loosely coupled but so is the entire agent device. This agent system is intended to produce a basic prototype agent device which manufacturers can integrate into an appliance, apply their role layer and then ship it as a single unit. If the manufacturer improves a layer, they can issue a new updated layer to devices since the agent roles are entirely software based. We describe the layers of Figure 1 in some detail in the following paragraphs.

Device Hardware The *Device Hardware* is at a basic level the processor, memory, storage and communications and a Linux operating system to manage it all. Linux grants the *Agent Core Layer* access the hardware (Linux handles hardware drivers). Any specialist hardware API(Application Programming Interface) for example a GPU or heating element. APIs are defined in either the *Agent+ Layer* if the special hardware is attached to the *Agent Device Unit* or in the *Agent Role Layer* if the special hardware is part of the appliance or utility which the agent device is attached to. For example if a coffee machine has an *Agent Device* attached to it, the *Agent Role Layer* would contain the APIs required for the *Agent Core* to use the coffee machine functionality. The *Agent Device* can be embedded within an appliance or attached to one. The hardware layer will provide a flag indicating to the agent whether it is embedded or attached.

Agent Core Layer As the name implies this layer is to be considered the core of the system. This layer will be generic across all implementations to combat system fragmentation[5] and maintain a high level of agent interoperability. This layer is an adaptable agent, adaptive in that is must work to complete tasks with any roles assigned to it. An agent engine will form the main part of this layer along with an array of layer APIs as well as a dynamic action cache used to formulate plans based on role and available abilities. The *Agent Core Layer* is not to be designed for dynamic updates to any functionality. Instead any functionality that requires an update is to be overloaded in the Agent+ layer. If an update within this layer is absolutely necessary (like security) then the agent device will require a restart unlike updating in any other layer. The Common Agent Engine provided by the *Agent Core Layer* creates by default at least one agent upon system start. This agent is considered the *Master Agent* of the system and takes on any default roles assigned to it. *Slave Agents* can be created to handle any addition

[5] System fragmentation refers to parts of a (common) system being incompatible with other parts. E.g., Apple iOS devices have low levels of fragmentation, since an app written for an iPhone 3 will work on an iPhone 5 and (mostly) vice versa. Compare this with the Android platform where there are huge differences, such that apps will generally come with extensive lists of supported devices/configurations. For our framework, this means that agents should be able to work together regardless of the developer, manufacturer, or purpose.

roles assigned to the agent system. The *Master Agent* has full control of the system hardware and *Slave Agents*. In doing so the *Master Agent* will have more responsibilities attributed to it compared to the *Slave Agents* who will only have to satisfy their role requirements.

Each agent has access to the *Agent+ Layer* and assigned roles from the *Agent Role Layer*. *Slave Agents* will have access to the hardware but the *Master Agent* will have assigned a 'Hardware Need Value' to each agent dependant on their role. This value represents a proportion of system resource usage and on the system adds up to 1. Consider a TV with access to a GPU and TV Tuner hardware and two agents currently running on it. One *Slave Agent* is tasked with recording a user's favourite shows and this has a GPU need of 0.2 and a Tuner need of 0.5. The other agent is the *Master Agent* for the TV system and is responsible for displaying content, programming recordings, and responding to user interaction with a GPU need of 0.8 and a Tuner need of 0.5 (assuming the user is watching live TV). With the TV off, the *Master Agent* allocates almost all GPU and Tuner usage to the *Slave Agent* based of its needs. This is a simplistic view of how resources could be managed and is intended to demonstrate one of the extra roles (resource management) the *Master Agent* will be assigned and how this may work.

Agent+ Layer This is an enhancement layer to the *Agent Core* designed to provide dynamic updates and extend core functionality when required. When an update is applied to the *Agent+ Layer* the *Agent Device* does not need to power down. The updating layer simply becomes locked while the update occurs and then – once successful – returns to an operational state. For instance, assume an *Agent Device* has a GPU(Graphics Processing Unit) and this *Agent Device* is attached to a fridge. The fridge will not have any need for a GPU so the *Agent Role Layer* will contain no API's for a GPU. Therefore there must be a way in which *Agent Device* developers can add any functionality directly to the *Agent Device* without having to create pseudo role for it. The *Agent+ Layer* allows for this kind of extension of core functionality.

Agent Role Layer Role specific functionality and APIs are stored here. The *Agent Role Layer* and *Agent+ Layer* are where agent developers will spend their time as these layers define what the agent is and how it should behave. If the agent role requires a certain type of hardware or proprietary algorithm to work as intended this would be implemented in the *Agent Role Layer*. An agent can have any number n of roles ($n \in \mathbb{N}$ can be 0) provided the hardware can support that number of roles. This can allow for agent-network-wide load balancing as roles can be duplicated to a numerous compatible *Agent Devices*. An Agents' compatibility for a role is dependent on hardware requirements for the role. An agent can accept a role if specialist hardware is not available to it, in this sense the agent takes on a support role. For example a security system might be trying to identify who is in the house. The Home PC is not in use and neither is the coffee machine so the security agent asks them to take on a support role and help process some of the data. The two devices are both in possession of a CPU capable of processing the data required by the security agent and therefore are compatible for the support role. Once the support role is no longer required, agents can make the decision to drop any extra roles[6]).

[6]Security agents can force agents to drop certain roles like security roles.

Types of Agents using the Layered Agent Framework Agents using the Layered Agent Framework come in three forms:

Standalone device (Agent Device) The agent is not attached to any appliance or utility. This type of agent acts as an agent controller for a specific area of the agent network, or a worker agent which can be used by other agents on the network for data processing much like a server.

Attached to an appliance/utility (Attached Agent Device) The agent is attached to an appliance and is enabled to use the functionality of the appliance. The actual agent device is a separate unit to the appliance. This setup is designed to allow current appliances and utilities to be adapted for agent technology by interfacing the agent unit with the appliance or utility. This also allows the agent unit to be removed for repairs or to disable the agent capabilities and control over an appliance.

Embedded in the appliance/utility (Embedded Agent Device) These agent devices will be part of the appliance/utility and cannot be removed. If a hardware fault occurs the system will default back to a 'dumb' unit until repairs are carried out.

3.2 Executable Model

We introduce a Petri net model based on the nets-within-nets paradigm. It builds on the MULAN multi-agent architecture and is implemented in the RENEW tool. We give a brief overview of MULAN in Section 3.2 and then introduce our model in Section 3.2.

Multi-Agent Petri Nets We focus on the MULAN architecture shown in Figure 2 as introduced in [11]. MULAN separates the multi-agent system into four parts or layers. The layers are: (a) agent network, (b) agent platform, (c) agent, and (d) protocol.

Each layer is represented by its own Petri net(s). Protocols specify the agent programs. Agents reside on a platform that provides internal and external communication facilities and is located in an environment within the multi-agent network of the multi-agent system. The latter determines the communication structure available to agents.

Because it is unrealistic to assume legacy products to (fully) support agent communication, we generalise parts of the MULAN architecture to reflect this scenario. In particular, the *agent platform* will become simply a *platform* and the *multi-agent network* will be referred to as *network*.

Multi-Agent Petri Net Components The Multi-Agent Petri Net Components (MAPNCs) run on a MULAN-based architecture. They constitute the building blocks of agents and protocols, e.g. for agents created at runtime by other agents. MAPNCs are presently limited in that their 'template' or 'blueprint' has to be defined prior to run time and have to have certain properties discussed below.

The possible templates are stored in a place of the agent net similar to the protocols. Whenever an instance is required, a copy of one of these templates is initialised according to its task and the initialised copy is then moved onto the platform from on which it operates. As a mobile agent, the generated instance (of an agent) can then traverse the network to reach other platforms and interact with (agent) nets at remote locations.

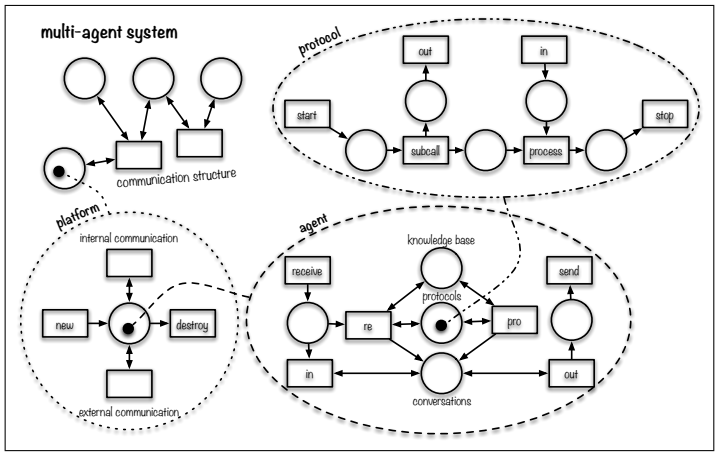

Fig. 2. MULAN agent architecture [11]

Figure 3 shows the structure of the extended MULAN architecture that includes the layer structure introduced in Section 3.1.

The extended MULAN provides the multi-layer support for agent communication by providing the layer functionality in an additional place regulating the communication by forcing synchronisation according to the layer information and allowing only communication based on the specifications therein. Communication can be 'negotiated' to take place on different levels according to the different layers.

If a method is overridden, this will result in its protocol becoming unavailable for execution in the Petri net model.[7]

3.3 Towards an Implementation

This section discusses some issues related to the implementation of our framework.

Brief Roadmap Designing any system requires an awareness of implementation. With this in mind any design decisions must made with consideration towards aspects like available system resources and response times. Our aim is to provide an architecture for flexible agent-based, object-oriented systems (Layered Agent Framework, LAF). The Agent Core represents the main component that every system using the LAF will be based on. Elementary communication features and functionality will be implemented on the first prototype system. At least one Agent+ layer will have to be created for testing with the Agent Core. This basic system set up will then be used to test the dynamic update capability of the framework as well as simple extensions to the Agent Core

[7] This is easily achieved by removal of a required resource, i.e. an input token to the respective transition and appropriate transition synchronisation. This is not shown explicitly in the extended model in Figure3, because it is implemented at a protocol net level and the relevant inscription detail had to be omitted for brevity and readability.

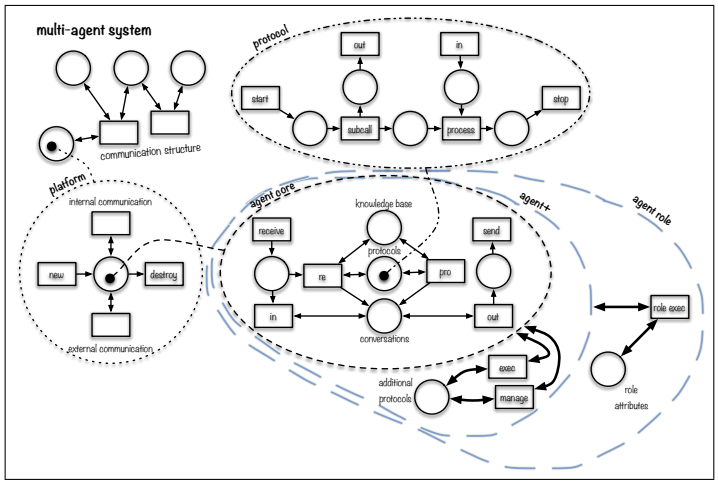

Fig. 3. Extended MULAN agent architecture

functionality. Once the Agent+ Layer has been implemented and tested, Agent Role Layers can be added. An arbitrary number of Agent Role Layers can be developed and deployed to make full use of the flexible design of the Layered Agent Framework. The details of what extensions, roles or testing will be done has not yet been determined. The framework will be supported by a set of development tools that will assist the developer with the development of bespoke solutions, i.e., Agent+ layers and agent roles. The tools will include templates and components to assist the construction of agent plans, for example they will supply action templates and agent creation blueprints.

Manual Override Every system build using LAF must have a manual override in case of user preference or fault, *"open the pod bay doors HAL"* [8]. This manual override will place the agent into disabled state in which it has no control over its appliance or utility. Such agents can still communicate their state on the network and attempt to assist other agents(assuming no fault). The manual override simply (in software terms) blocks the agents' control over its appliance/utility which from then on will need to be user operated. The agent can be 'enabled' by the user flipping a switch or by asking the agent network to enable the agent again(expressed consent must be given).

The Challenge of Real-Time Responsiveness There are many challenges facing the final implemented framework and any system based on it. Most importantly, the system must work in real time. Having a system working in a home environment in which the user must wait more than a minute for a response is generally unacceptable. A user can expect delays in response for processing actions such as *"What times are the buses to Cardiff"*, *"one moment.... 1315 and each hour from then till 2100"* this kind of scenario

[8] Stanley Kubricks' *2001 A Space Odyssey*. HAL is asked to open the pod bay doors, HAL refuses this request with the response *"I'm sorry Dave, I'm afraid i can't do that"*.

is acceptable. However if the user asked for the curtain to be drawn and had a 2 minute wait before the curtain even started to move is very undesirable. This also poses the question of user feedback when there are unavoidable delays, but this is outside the scope of this paper, since it would be the programmer's duty to supply the feedback.[9] Aside from real-time responsiveness the system must have a high fault tolerance and some form of system crash recovery. A computer crashing is one thing but an entire home of systems crashing could be disastrous and dangerous. Hence, we need systems with the ability to recover quickly from a major fault or system-wide disruption. Any solutions must attempt not to affect the user in a noticeable way. The perfect recovery solution would be the system completely recovering from a system wide fault without the user knowing there was ever anything wrong.

The Challenge of Agent Security Security is a major concern in the home, especially considering the amount of personal data that will be stored on the users' lives. There are many ways in which data can be protected, ranging from passwords to strong data encryption. As security measures placed upon data are increased the less dynamic and freely available that data becomes and a balance needs to be negotiated such that some data become accessible without compromising the required level of data protection. One mechanism could be a relational notion of trust that agents can have with other agents. The idea is that agents who have proven they are not a threat to the system over time will become trusted by other agents. An untrusted agents' request may be rejected, in contrast trusted agents are more likely to have requests completed by other agents. Certain agents can be pre-set by developers to have a trust limit for example security systems should only have complete trust in other home security components with a certified ID. A private trust value held by every agent on every other agent allows new devices to use the network without compromising on the security of the network. Trust levels would range from 0 to 1, representing the range from *untrusted* to *fully trusted*. A new device will usually start on a neutral trust level of 0.5 (unless otherwise pre-set). Unless the new device was a permanent addition to the smart network (e.g., a cooker) for which exceptions can be made, if permitted by the security system. Trust values will be updated dynamically at run time and can be influenced by the security system.

4 Conclusion

We have introduced an agent architecture that supports object-oriented concepts most software developers are familiar with and that adds component-based agent layers. These layers provide basic and extended agent reasoning and communications facilities in a maximally flexible way. The architecture is primarily targeted at smart home automation. For this, it supports dynamic reconfiguration and seamless integration with non-agent-based systems. This is required because the user experience is of foremost importance in the smart home application area. Easily configurable components can be added at runtime to provide additional features and to configure security features as required by individual sub-systems.

[9] The tools to be developed for the framework will have templates for a variety of devices and appliances that can be equipped with basic user feedback functions.

A Petri net implementation has been presented in this paper. The next steps will be to implement the framework on inexpensive hardware that can be integrated into new appliances or added to existing devices, e.g. in the form of a simple wall-plug adapter.

Also following successful testing of the framework, will be the provision of a tool kit with components and design patterns to ease the construction of agents, layers, and roles. Ideally, we would like to link the visual creation of the latter (by means of agent-based object Petri nets) to the development environment, providing an automated translation of the Petri-net representation into an agent program. The Petri net model could then be analysed using existing methods and tools, while the properties are preserved by a verified translation procedure.

References

1. Köhler, M., Farwer, B.: Modelling global and local name spaces for mobile agents using object nets. Fundamenta Informaticae **72**(1-3) (2006) 109–122
2. Ceccaroni, L., Verdaguer, X.: Agent-oriented, multimedia, interactive services in home automation
3. Fraile, J., Bajo, J., Lancho, B., Sanz, E.: Hoca home care multi-agent architecture. In Corchado, J., Rodríguez, S., Llinas, J., Molina, J., eds.: International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008). Volume 50 of Advances in Soft Computing. Springer Berlin Heidelberg (2009) 52–61
4. Sriskanthan, N., Tan, F., Karande, A.: Bluetooth based home automation system. Microprocessors and Microsystems **26**(6) (2002) 281–289
5. Blackwell, A.F., Rode, J.A., Toye, E.F.: How do we program the home? gender, attention investment, and the psychology of programming at home. International Journal of Human-Computer Studies **67**(4) (2009) 324 – 341
6. Nunes, R.J.C.: Home automation - a step towards better energy management. IST – Technical University of Lisbon / INESC-ID R. Alves Redol, 9, 1000-029 Lisboa, Portugal (2003)
7. Abras, S., Ploix, S., Pesty, S., Jacomino, M.: A multi-agent home automation system for power management. In Andrade-Cetto, J., Ferrier, J.L., Pereira, J.D., Filipe, J., eds.: ICINCO-ICSO, INSTICC Press (2006) 3–8
8. Giordana, A., Mendola, D., Monfrecola, D., Moio, A.: Horus: an agent system for home automation. In: 13th Workshop on Objects and Agents (WOA 2012), CEUR Workshop Proceedings Vol-892 (ISSN 1613-0073) (2012)
9. Dennis, L., Farwer, B., Bordini, R., Fisher, M., Wooldridge, M.: A common semantic basis for BDI languages. In Dastani, M., Seghrouchni, A.E.F., Ricci, A., Winikoff, M., eds.: Proceedings of the International Workshop on Programming Multi-Agent Systems (ProMAS 2007). (May 2007) 88–103
10. R. H. Bordini, L. A. Dennis, B.F., Fisher, M.: Directions for agent model checking. In M. Dastani, K. V. Hindriks, J.J.C.M., ed.: Specification and Verification of Multi-agent Systems. Springer US (2010) 103–123
11. Köhler, M., Moldt, D., Rölke, H.: Modelling the structure and behaviour of petri net agents. In Colom, J.M., Koutny, M., eds.: Applications and Theory of Petri Nets 2001. Volume 2075 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2001) 224–241