RoDaFlow: A Framework for Development of Dataflow Network Agents in Smart-M3 with Substitution Method

Denis Laure, Ilya Paramonov
P. G. Demidov Yaroslavl State University
Yaroslavl, Russia
den.a.laure@gmail.com, ilya.paramonov@fruct.org

***Abstract*—The paper describes RoDaFlow—a framework for development of dataflow network agents on Smart-M3 platform. The agents created with the use of the framework support the substitution mechanism that allows to keep the dataflow network based systems in working conditions when some of its agents fall out. The RoDaFlow framework is implemented in Java with the use of Smart-M3 Java KPI library. It allows to develop primary and substitute agents by implementing only their programs, which define how agents process incoming information. The paper also describes a prototype of home light control system based on the dataflow network that allows to effectively control the light in a home. This system uses a number of sensors, actuators, remote control units and computational agents. The agents use information from remote control units and sensors to control the actuators and thus the light level in the home rooms. The agents for the prototype were developed with the use of proposed framework.**

Keywords*—*Framework, Dataflow network, Agent substitution, Smart-M3.

I. Introduction

The dataflow network is an architecture for distributed computing systems [1]. It consists of computational units called nodes. Each node retrieves information from some other nodes, processes it and forwards further to next layers of the network. This activity forms the flow of the data that passes from an information source (for example, sensor) through a number of computational nodes to the network unit that uses processed data (for instance, to control some physical entities or display the data).

In some cases the computational node can be disconnected from the dataflow network, for example due to breakage of the device that hosts this node. Such situation may lead to the rupture of the dataflow that results in the whole system disfunction.

As a result of our previous researches we developed the substitution mechanism for dataflow networks [2]. Its implementation based on Smart-M3 platform [3]. The goal of the mechanism is to prevent dataflow disruption and to keep system based on dataflow network in working conditions even when some of its nodes were disconnected.

The main idea of the substitution mechanism is that the dataflow network along with the usual computational agents also consists of agents of special type, called substitute agents. In case when primary agent disconnects from the network, because of connection loss or lack of the energy, it is being substituted by the substitute agent. This agent performs same or almost the same computations and does not allow the dataflow to be ruptured.

In this paper we introduce RoDaFlow—a framework for creating primary and substitute agents for dataflow network implementation on Smart-M3 platform. As the behavior and basic operations of the agents are always the same, there is no need to implement them each time again. The framework allows to create agents by implementing only agent programs. The program determines what type of information is processed by the agent and the way of this processing. It is the only thing that differs between the agents.

The rest of the paper is organized as following. In Section II we describe the aspects of primary and substitute agents implementation in Smart-M3. Section III contains a detailed description of the framework architecture. Section IV gives a closer look on the process of creating agents with the use of the introduced framework. In Section V we describe the prototype of the home light control system, which agents were developed with the use of the framework. Section VI concludes the paper.

II. Dataflow Network Agents Implementation in Smart-M3

This section briefly overview common behavior of primary and substitute agents, which was described in our previous papers including [2].

In the dataflow network implementation on Smart-M3 platform agents correspond to knowledge processors (KPs). They subscribe to input triples and produce output and state triples as the input updates. The implementation of primary and substitute agents in Smart-M3 differs and is described below.

A. Primary Agent

KP initializes as the primary agent by inserting a set of description triples into data storage after joining the network. This triples with their description are shown in Table I. All the triples have the same subject that is the id of the primary agent. The strings in apostrophes are the values of the certain triple components.

TABLE I. PRIMARY AGENT DESCRIPTION TRIPLES

Triple subject	Triple predicate	Triple object	Triple description
primary_agent_id	*'Type'*	*'DataflowAgent'*	defines that current agent is a node of the dataflow network
	'Description'	*URI*	determines URI with the description of the agent's operations
	'SubstituteProgramType'	*program_type*	defines the type of the substitute program. The explanation of the meaning of this triple is given in section II-B
	'SubstituteProgram'	*substitute_program*	defines the code of the program that will be performed by the substitute agent during the substitution of the current agent
	'Active'	*active*	defines whether the agent is active or not. The *active* value can be either 'yes' or 'no'

The substitute program consists of two parts: header and body. Program header contains three sets of triples' templates for substitute agent's input, output and state triples. The body of the program defines what calculations the substitute agent performs.

The *(agent_id, 'Active', active)* triple is used by Semantic Information Broker (SIB) to determine whether the agent is able to work. Active agent performs its operations as usual. If the agent is deactivated, then it is not functioning. Agent deactivation can be performed in two cases. The first one is when the SIB detects breakage of agent's connection. In the other case the agent can deactivate itself right before it disconnects from the network. For example, if the battery level of the device, which runs the agent, is low and it is needed to stop all agent calculations to save the energy. When the agent returns to the network after disconnection, it activates itself. The activation/deactivation of the agent leads to start/stop of its substitution correspondingly.

After the initialization the KP subscribes for its input triples. Each time the input is updated the KP recalculates the output and state triples.

B. Substitute Agent

Description triples of the substitute agent differ from primary agent ones. They are shown in Table II.

After initialization the KP subscribes for its *'Substitutes'* triple and starts waiting for its changes. As the SIB detects the failure of some primary agent, it searches for the appropriate (i.e., with the same program type) substitute agent. The SIB changes the object of the found agent *'Substitutes'* triple from 'None' to the id of the broken primary agent. It is a signal for the substitute agent to start substitution. The substitute agent retrieves the substitute program from the primary agent description triple, subscribes for input triples stated in there and then activates itself. Each time the input changes, the substitute agent executes the program to calculate new output and state triples.

If the substituted primary agent returns to the dataflow network, the SIB changes the object of the *'Substitutes'* triple back to 'None'. The substitute agent notices it and stops the substitution. It unsubscribes from all input triples, deactivates itself and starts waiting for another call for substitution.

The behavior of primary and substitute agents, which is described above, is always the same. This fact motivated us to create a framework that contains all the general logic of both types of agents.

III. FRAMEWORK ARCHITECTURE

We implemented the RoDaFlow framework in Java. It uses Smart-M3 Java KPI library [4] for communication with the smart space. The class diagram of the framework is shown in Fig. 1.

The RDF triple entity is represented in the framework with the `Triple` class. It stores triple's subject, predicate, object and types of subject and object. It also contains getters and setters for these triple components. The `Triple`'s `toVector` method converts triple to a form of `Vector` of `String`s. Such form of triple is used in Smart-M3 Java KPI library.

The `Triple` class also allows to create triple templates. It is done by setting triple subject and/or object to the `null`, which means that this component is arbitrary. Another methods of the `Triple` class – `equalsToTemplate` and `equalsToOneOfTemplates` – allow to match the triple against one or more templates correspondingly. The former method returns true if the triple matches given template and false otherwise. The latter one returns the first template from the given list of the templates, which is matched by the triple.

The `TriplesCouple` class represents a container of two triples. These triples represent the new and the obsolete values of the triple that is updated in the smart space.

The core of the framework is the group of four agent classes. The first on is the `Agent` class. It is a layer between the framework and Smart-M3 Java KPI library. It provides basic operations to smart space (query, insert, remove, update, subscribe and unsubscribe) with the use of triple representation as an instance of the `Triple` class. In addition this class also provides methods to setup and remove triples protection [5]. This protection is used to restrict changing of agent description, its output and state triples by other dataflow network nodes.

Any smart space operation can fail due to various reasons (e.g., agent disconnection from the smart space, attempt to change protected data, etc.). To handle such errors the instance of the `OnAgentErrorListener` interface can be set for the `Agent`. The `onAgentError` method of the interface is called each time the error occurs during the operations.

The `DataflowAgent` class inherits the `Agent` class and represents agent of the dataflow network. It provides common operations and properties of these agents, such as agent id and URI that are used to identificate the agent. This class allows to activate and deactivate the agent by calling corresponding methods and to handle incoming subscription messages. The `DataflowAgent` class has abstract `handleSubscription` method that is called in a separate thread each time the agent receives subscription message.

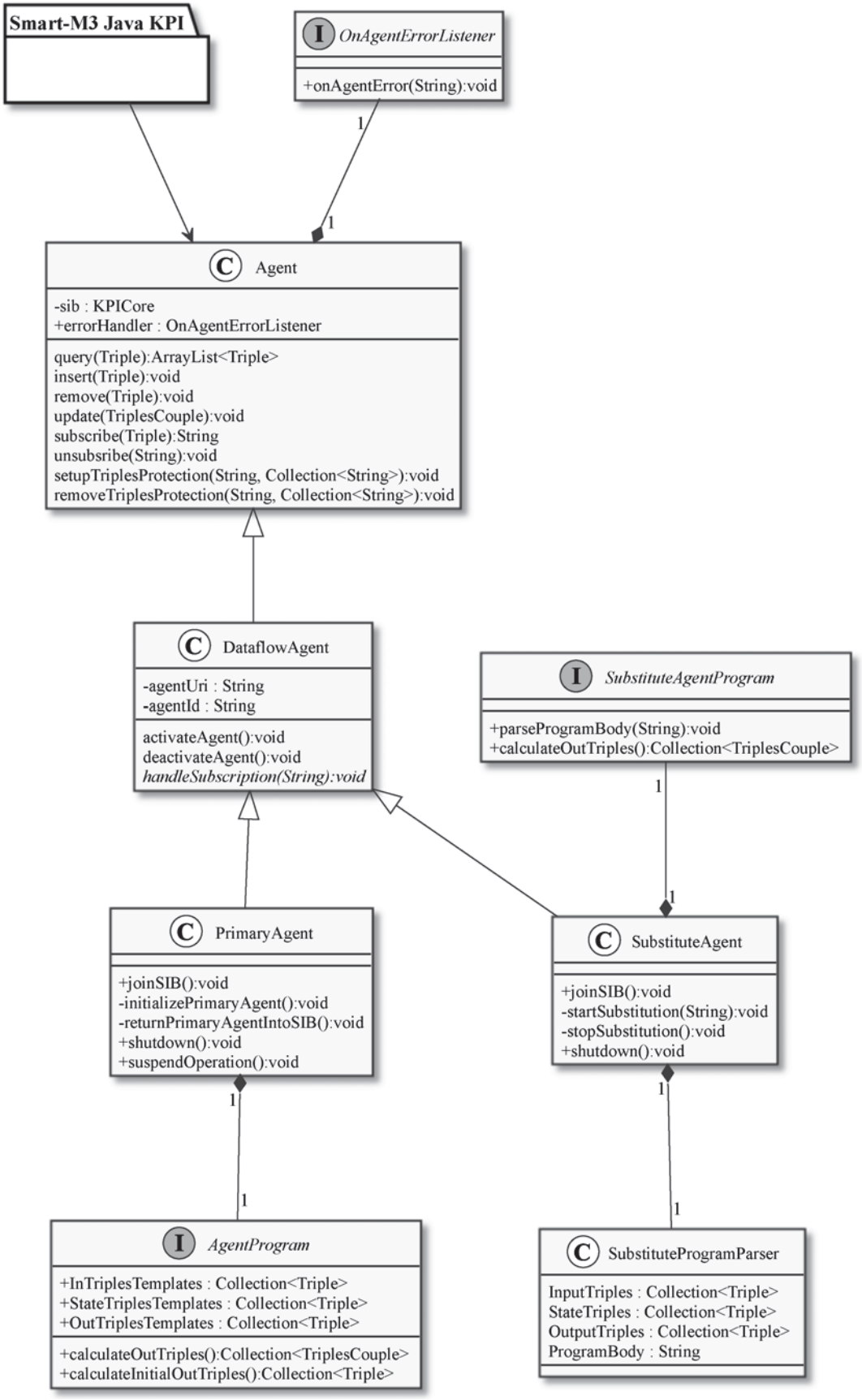

Fig. 1. Framework class diagram

TABLE II. SUBSTITUTE AGENT DESCRIPTION TRIPLES

Triple subject	Triple predicate	Triple object	Triple description
substitute_agent_id	*'Type'*	*'DataflowAgent'*	defines that current agent is a node of the dataflow network
	'SubstituteProgramType'	*program_type*	defines what type of substitute program this agent can run. The substitute agent is able to substitute the primary agent only in case their substitute program types are equal
	'Active'	*'no'*	defines whether the agent is active or not. Substitute agent initially is deactivated
	'Substitutes'	*'None'*	defines what agent is substituted by this one

The `PrimaryAgent` and the `SubstituteAgent` classes inherit the `DataflowAgent` class and represent primary and substitute agents correspondingly. These classes are used by the developer to create dataflow agents that support substitution mechanism. More closely the process of creation of agents is described in the following section.

IV. AGENTS IMPLEMENTATION

The process of agents implementation starts with the defining of agent's programs. For this purpose there are the `AgentProgram` and the `SubstituteAgentProgram` interfaces in the framework (see Fig. 1).

The `AgentProgram` interface represents a program for the primary agent. To implement this interface one should define output, input and state triples for the agent program. It is done by implementing methods that return list of corresponding triples templates. Input triples are used as incoming information for calculation of output triples. State triples describe current calculation status of the program. Another two methods that are defined by the `AgentProgram` interface are `calculateInitialOutTriples` and `calculateOutTriples`. The first one is used to calculate initial state and output triples using input triples. It is called only once, when the primary agent starts its work. The second method is called each time one of the input triples is updated in smart space by another agent and calculate new output and state triples.

The `SubstituteAgentProgram` interface represents a program for the substitute agent. This program performs the calculations that depend on the substitute program code, which is received by the substitute agent. According to this fact the `SubstituteAgentProgram` interface defines two methods: `parseProgramBody` and `calculateOutTriples`. First one is called each time the substitute agent starts substitution, when it receives the code of the substitute program. The method parses the body of the received substitute program and determines what will `calculateOutTriples` method do. The last one performs calculations each time some of the input triples, pointed in the substitute program header, is updated.

After implementation of the agent program developer just needs to create an instance of the `PrimaryAgent` or the `SubstituteAgent` class (see Fig. 1) passing implemented program and other additional information to it and call its `joinSIB` method. After that the agent automatically starts its work. The primary agent:

1) joins the smart space;
2) initializes itself inserting description triples;
3) inserts protection triples for its description, output and state triples;
4) calculates and inserts initial output and state triples;
5) subscribes to its input triples.

If this primary agent (the primary agent with the same id) already was in smart space, but was disconnected from it by some reason (for instance, due to lack of the energy), then during the initialization it only subscribes to input triples. After the initialization the primary agent will calculate new output and state triples each time it receives subscribe message about input triples update from the smart space.

After calling the `joinSIB` method on the `SubstituteAgent` class instance it:

1) joins the smart space;
2) initializes itself inserting description triples;
3) inserts protection triples for its description triples;
4) subscribes to triple that indicates what primary agent is substituted by this one.

As the substitute agent receives subscription message with the id of the agent it must substitute, it starts the substitution. This operation consists of following steps:

1) query from smart space triple with the substitute program code and parse this code using `SubstituteProgramParser` class;
2) query from smart space input, output and state triples;
3) pass the body of the substitute program to the `SubstituteAgentProgram` instance;
4) subscribe to input triples;
5) activate agent.

Both the `PrimaryAgent` and the `SubstituteAgent` classes contain the `shutdown` method that can be used by the developer to finish agent work. This method:

1) removes all the input, state and description triples of the agent;
2) removes all protection triples, which were added by the agent;
3) leaves smart space.

In addition, the `PrimaryAgent` class contains the `suspendOperation` method that deactivates the agent provoking it to be substituted. It can be used by the developer to:

- test the work of the substitution and substitute agents;
- preventively deactivate the agent before it disconnects from the smart space for some reason.

Therefore, to create developer's own agent he or she needs to implement one of the agent program interfaces, create instance of the agent class, pass this program to the agent and call agent's `joinSIB` method. So the creation process is simple and does not require from the developer to implement operations that are equal for all agents. Moreover, the developer is not communicating straight with the smart

space. He or she just defines what triples the agent uses and what calculations it performs.

In such a way it saves the developer's time and simplifies the development of the agents and hence the development of dataflow network based systems. The framework also does not require from the developer any additional knowledges of substitution mechanism implementation to create agents that support this mechanism.

V. Case Study

In this section we describe the prototype of the home light control system based on dataflow network implementation for Smart-M3 platform. It can be used as part of the smart home along with other systems [6], [7]. The system is designed to remotely control the light in the whole home or apartment. Its primary and substitute agents are developed with the use of the proposed framework. The example of the system dataflows in one room is shown in Fig. 2.

Home light control system consists of following components:

- indoor illuminance sensors in each house room;
- outdoor illuminance sensors on each house window;
- sensors that count the number of people that entered or left the room (in each room entrance);
- actuators that manage blinds level (i.e., control the amount of light comes through the windows) on each home window;
- actuators that set the brightness of each lamp or other electrical light source;
- remote control unit, with which the user can set the preferable light level in each house room;
- agent that uses information from room sensors and remote control unit to control the actuators in particular room (one agent per room);
- substitute agents for each of the system's primary agents.

The system prototype works as follows. The user sets the preferable light level in each room using remote control unit. The primary agents in each room receive information from the room sensors and remote control unit and set the optimal parameters for the room actuators. Parameters that allow the electricity consumption of the system to be the minimal possible considered as the optimal. If there are people in the room, then the parameters applied immediately. Otherwise the parameters are applied only after someone enters the room. If everybody, who was in the room, left it and the room stays empty for 5 minutes then the light in the room is automatically switched off. The light is automatically switched on as someone enters the room.

As a primary agent falls out, the substitute agent starts to work. It provides the same functionality as the substituted agent, but it cannot control the blinds actuators. In this case the user is assumed to control the level of the blinds as he or she wishes.

The system prototype uses the following triples:

- Each illuminance sensor generates the triple that contains sensor id and the measured value.
- Each room entrance sensor generates the triple that contains sensor id and the number of the people inside room.
- The remote control unit generates triples that contain the id of the room and user set value of the illuminance in this room.
- Each actuator subscribes for the triple that contains its id and the parameters for this actuator.

Each primary agent subscribes to the triples generated by the sensors in the room and also to the triple that contains user set light level in the room. The room actuators subscribe to the triples that are output of room's agent.

The framework allowed us to not implement the whole agent logic including initialization, subscription handling, shutdown, etc. All we had to implement were just the agent programs. The implementation of such program consists of:

- definition of output and input agent triples sets, which were described above;
- implementation of the calculation method that uses input triples to determine the output triples.

This case study shows that the RoDaFlow framework allows to save the time and significantly simplify the development of dataflow network based systems.

VI. Conclusion

The RoDaFlow framework was implemented in Java and used to develop agents for the home light control system prototype. The framework allows to create agents for the systems based on dataflow network implementation in Smart-M3. These agents support substitution mechanism described in [2], which allows to make dataflow networks more robust. The creation of the agents is very simple and requires from the developer to implement only the programs for the agents. The RoDaFlow framework can be downloaded from its home page: *https://yar.fruct.org/projects/rodaflow*.

As the framework is written in Java it allows to create agents for all popular desktop platforms, such as Windows, Linux and Mac OS. Moreover, it is also possible to use the proposed framework for creating native mobile agents for Android devices, which currently have about 79 percent of the whole smartphones market share [8]. In future the RoDaFlow framework can be used to create dataflow agents for devices based on Oracle's Internet of Things platform [9].

Acknowledgment

The study was supported by The Ministry of education and science of Russia, project 14.B37.21.0876. The authors would like to thank Andrey Vasilev for his valuable feedback and fruitful discussions.

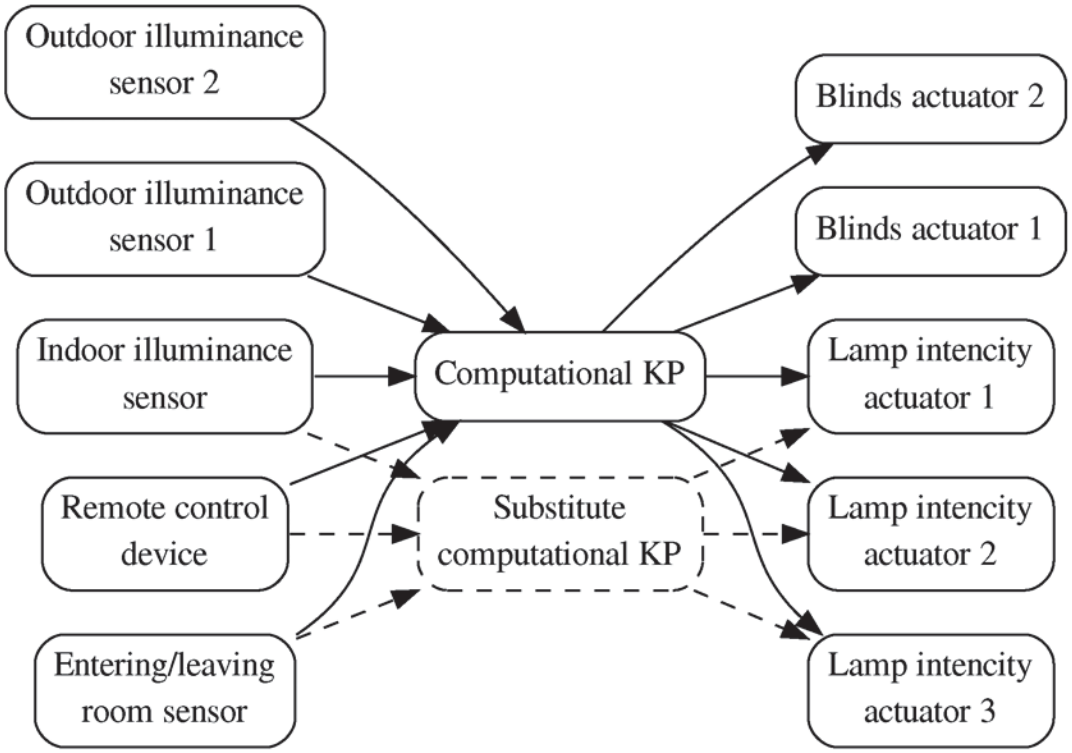

Fig. 2. The example of the light control system dataflows for one room

REFERENCES

[1] J. W. Janneck, I. D. Miller, D. B. Parlour, G. Roquier, M. Wipliez, and M. Raulet, "Synthesizing hardware from dataflow programs," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 241–249, 2011.

[2] I. Paramonov, A. Vasilev, D. Laure, and I. Timofeev, "Agent substitution mechanism for dataflow networks: Case study and implementation in smart-m3," in *Internet of Things, Smart Spaces, and Next Generation Networking*, ser. Lecture Notes in Computer Science, S. Balandin, S. Andreev, and Y. Koucheryavy, Eds. Springer Berlin Heidelberg, 2013, vol. 8121, pp. 60–71.

[3] J. Honkola, H. Laine, R. Brown, and O. Tyrkkö, "Smart-M3 information sharing platform," in *IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2010, pp. 1041–1046.

[4] (2013, April) Smart-M3 Java KPI library. [Online]. Available: http://sourceforge.net/projects/smartm3-javakpi/

[5] A. DElia, J. Honkola, D. Manzaroli, and T. S. Cinotti, "Access control at triple level: Specification and enforcement of a simple RDF model to support concurrent applications in smart environments," in *Smart Spaces and Next Generation Wired/Wireless Networking*. Springer Berlin Heidelberg, 2011, pp. 63–74.

[6] J.-h. Song and S.-f. Hou, "Infrared application in smart home system-based on intelligent air conditioning design," in *Proceedings of 3rd International Asia Conference on Industrial Engineering and Management Innovation (IEMI2012)*, R. Dou, Ed. Springer Berlin Heidelberg, 2013, pp. 721–728.

[7] M. Jahn, M. Jentsch, C. R. Prause, F. Pramudianto, A. Al-Akkad, and R. Reiners, "The energy aware smart home," in *5th International Conference on Future Information Technology (FutureTech)*, 2010, pp. 1–8.

[8] (2013, August) Gartner says smartphone sales grew 46.5 percent in second quarter of 2013 and exceeded feature phone sales for first time. [Online]. Available: http://www.gartner.com/newsroom/id/2573415

[9] (2013, October) Oracles internet of things platform. [Online]. Available: http://www.oracle.com/us/solutions/internetofthings/index.html