Enabling the business-based
Internet of Things and Services

(FP7 257852)

D5.2.3 Architecture for intelligence integration 3

Published by the ebbits Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme
Objective ICT-2009.1.3: Internet of Things and Enterprise environments

Document version: 1.1 Submission date: May, 27 2013

Document control page

Document file: D5.2.3 Architecture for intelligence integration 3.docx
Document version: 1.1
Document owner: Ferry Pramudianto (FIT)

Work package: WP5 – Distributed and centralized intelligence
Task: T5.1 – Validation of platform and services
Deliverable type: R

Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Ferry Pramudianto	2013-1-15	Context Model Reference, Context modeling tool architecture
0.2	Andreas Zimmermann	2013-1-15	Introduction to personalization
0.3	K. Furdik (IS)	2013-1-15	Conceptual representation of context in IoT applications
1.0	Ferry Pramudianto	2013-2-28	Final version submitted to the European Commission
1.1	Ferry Pramudianto	2013-5-27	Removed the not-directly-relevant related work as requested by the commission Restructure chapter 5 for better readability. Checked and Fixed the figure references

Internal review history:

Reviewed by	Date	Summary of comments
P. Kostelnik (TUK)	2013-02-27	Approved with comments
K. Furdik (IS)	2013-02-27	Approved with comments

Index:

1 Executive Summary

This deliverable presents a continuation of D5.2.2 which discussed the architecture of the context manager and self-* properties. In this report, we present the list of open requirements that are relevant for WP5, as well as the requirements that are particularly being addressed in this deliverable.
Next in Chapter 3, a reference context model for the internet of things, people, and services is presented. This model can be extended by developers who are aiming at building applications on top of ebbits platform. The reference model may also serve as a guide for developers in deciding which parameters can be used to adapt the behavior of their systems. Moreover, this deliverable also discusses the architecture for ebbits context modeling tool which is designed to help developers building a prototype application rapidly. The tool is designed as a model-driven graphical tool that takes advantage of a simplified domain-specific language which is easy to learn and thus is capable to reduce the learning curve needed by ebbits developers.
The tool will be implemented in the next deliverable "D5.4.3 Multi-sensory fusion and context awareness prototypes" at M34.

2 Introduction

2.1 Purpose, context and scope of this deliverable

This work is part of the Task 5.3 - Context modeling and Task T5.2 – Intelligent sensor fusion services. In this context, the deliverable updates the architecture specification in *D5.2.2 Architecture for intelligence integration 2.* The content of this deliverable is meant to describe a reference context model as a framework and guide for developers that intend to build context aware applications on top of ebbits.

Moreover this deliverable is intended to describe the architecture of the context aware tool to support ebbits developers in developing intelligent applications. The architecture presented in this deliverable includes the architecture for ebbits context modeling tool which is designed to help developers building a prototype application rapidly. The tool is designed as a model-driven graphical tool that takes advantage of a simplified domain-specific language which is easy to learn and thus is capable to reduce the learning curve needed by ebbits developers.

2.2 Background

The ebbits platform is a cloud based platform that allows the "Internet of People, Things and Services" (IoPTS) to be integrated into existing and new Enterprise systems thus allowing firms and organizations to launch applications that rely on legacy data as well as real-time information from the physical world.

The ebbits platform supports interoperable business applications with context-aware processing of data separated in time and space, information and real-world events (addressing tags, sensors and actuators as services), people and workflows (operator and maintenance crews), process management using high-level business rules (energy and cost performance criteria), end-to-end business processes (traceability, lifecycle management), or comprehensive consumer demands (product authentication, trustworthy information, and knowledge sharing).

It will provide semantic resolution to the Internet of Things and hence present a new bridge between backend enterprise applications, people, services and the physical world, using information generated by tags, sensors, and other devices and performing actions on the real world. The platform will be based on a Service-oriented Architecture (SoA), transforming every device into a service. The SoA will allow these services to semantically discover, configure, and communicate with each other.

The ebbits platform will be demonstrated in end-to-end business applications featuring connectivity to and online monitoring of a product during its entire lifecycle, i.e. from the early manufacturing stage to its end-of-life.

3 Open Requirements

Requirements and architecture influence each other. Requirements are taken as an input for the architectural design process since they frame the architectural problem and explicitly represent the stakeholders' needs and desires. On the other hand, during the architecture design the system developers have to take into considerations what is possible and look at the requirements from a risk/cost perspective.

In order to discuss the state of play and visions for WP5 with the users, several developer workshops were organized at FIT. These workshops provided an environment for ebbits developers and user partners to discuss ideas and questions, and consequently to come to a common understanding of visions and problems to be resolved for the context management in ebbits.

The aim of these procedures was the systematic formalization of all relevant stakeholder requirements for the context management functionality, mainly based on the results of two workshops held in FIT in June 2012. Input to the scenarios and the initial requirement elicitation phase also came from the technical meetings, DoW, and deliverables D2.5.1, D2.5.2, and D2.5.3 Prototype application specifications. The process of gathering, maintaining, updating, and fulfilling requirements, which was elaborated within WP2, has been applied for the intelligence integration architecture design and functional components development as well.

After extraction and elicitation of requirements, these were described and organized in a tool named JIRA[1]. JIRA is a web based bug tracker that allows implementing and tracking the workflow of the Volere schema[2]. All partners in ebbits have been given access to JIRA with a unique username and password so that they can create requirements. This tool is necessary to ensure that all important details and procedures in the Volere Schema are properly adhered.

The extraction of requirements was accomplished in accordance with the scenarios presented in (D2.1 Scenario for usage of ebbits platform (ebbits 2011)). The requirements, together with attributes such as summary description, priority, fit criterion and current status of implementation are listed in the following Table 1. The value in the first "Key" field is used as an ID for uniquely identify the requirement on the JIRA Platform.

Table 1. List of all open requirements for the WP5.

Key	Summary	Priority	Fit Criterion	Status
EBBITS-210	System should provide location tracking of context entities	Blocker	Location is available as a generic context attribute (EBBITS-330), and applications can specify which sensors to use for location tracking (and choose among different standard tracking methods)	Planned for M36 Demo
EBBITS-213	System should show Energy Cost for different granularity of production processes	Blocker	Each automated process, machine is able to show energy cost	Planned for M36 Demo
EBBITS-398	Java based object oriented context modeling	Blocker	developers can define context model in java	Planned for M36 Demo
EBBITS-328	Sensor fusion algorithms must be realized as a decoupled component.	Major	Sensor fusion algorithms are available as services or libraries to the entire platform.	Planned for M36 Demo

[1] www.atlassian.com/JIRA

[2] http://www.volere.co.uk/template.htm

Key	Summary	Priority	Fit Criterion	Status
EBBITS-200	Distributed data can be referenced in data fusion and context management	Major	References to remote data can be defined and the queries can be executed.	Planned for M48 Demo
EBBITS-326	The system should compensate deviations of incoming data.	Major	System provides configureable filter to exclude outliers e.g.: define upper & lower threshold	Planned for M36 Demo
EBBITS-246	Dynamically loaded libraries must undergo a security check before their usage	Major	Dynamically loaded libraries must contain a valid signature in order to prevent security breaches in the system.	Planned for M36 Demo
EBBITS-196	Diagnostic component to detect and correct malfunctions	Major	Malfunctions or strange behaviour of machinery are recognized early enough.	Planned for M36 Demo
EBBITS-332	Context management should be able to process a large number of sensor events	Major	Context management is able to process at least 500 events / second.	Planned for M36 Demo
EBBITS-309	ebbits platform should have a publish-subscribe system	Major	Directory of alerts/events. The ebbits system includes one or more directories of alerts or events, including for each item the list of subscribers.	Planned for M36 Demo
EBBITS-223	The system provides access to aggregated/selected information through filters or fusion	Major	Processes can specify that information should be fused or filtered, and they only get the requested information	Planned for M36 Demo
EBBITS-381	Self-* manager needs to monitor the connection to the physical devices	Major	Self-* manager implementation is available that can handle unstable device connection.	Planned for M36 Demo
EBBITS-382	Device proxies can shut off a physical device from the network if it causes a lot of problem	Major	Shut-off functionality available, logic to detect problems designed.	Planned for M36 Demo
EBBITS-384	Device proxies reset devices upon problems when no other fix is defined by the developer	Major	Shut-off functionality available, logic to detect problems designed.	Planned for M36 Demo
EBBITS-383	Device proxies adjust event publishing frequency according to the network bandwidth	Major	Control management services available.	Planned for M36 Demo
EBBITS-385	Device Proxies need a standardized interface that provides control management services for event publication	Major	Device proxy interface defined	Planned for M36 Demo
EBBITS-397	Prototyping tools for context modeling	Major	model driven tool for modeling context exists	Planned for M36 Demo
EBBITS-224	Early maintenance notification when needed	Minor	Show 3-5 early maintenance use cases. Users/technicians are notified if robots need maintenance	Planned for M48 Demo

Key	Summary	Priority	Fit Criterion	Status
EBBITS-333	Libraries must only be accessible only for permitted applications.	Minor	The dynamic loading of libraries must be restricted through policies.	Planned for M48 Demo
EBBITS-242	The system should be able to take decision based on uncertain facts	Minor	The system supports at least two different soft-logic algorithms. e.g.: Fuzzy logic & probabilistic approach	Planned for M48 Demo
EBBITS-399	Reference context model for Internet of things, people and services	Blocker	developers could extend or modify the reference context model if he's building his application	Planned for M36 Demo

After we conducted the feasibility analysis and the significance of innovations for ebbits, we concluded that at the moment we should focus on these requirements:

- EBBITS-399 Reference context model for Internet of things, people and services
- EBBITS-309 ebbits platform should have a publish-subscribe system
- EBBITS-223 The system provides access to aggregated/selected information through filters or fusion
- EBBITS-397 Prototyping tools for context modeling
- EBBITS-398 Java based object oriented context modeling
- EBBITS-328 Sensor fusion algorithms must be realized as a decoupled component.

4 User Context in ebbits

4.1 Conceptual representation of context in IoT applications

The context for adaptive applications has been captured and modeled in the last decade under the paradigm of ubiquitous or pervasive computing (Satyanarayanan, 2001), which corresponds to the Internet of Things, People, and Services notion in terms of adaptable interactions between people and devices in a computer-enabled environment (Poslad, 2009). The context-aware behavior in such an environment is based on a context information and related reasoning of spatial, temporal, or other similar quantitative or qualitative characteristics of actors interacting within the IoT system. To overcome the heterogeneity of various application-specific definitions and consequent models of the context information, (Topcu, 2011) proposes a generic classification of different context types presented in Figure 1.

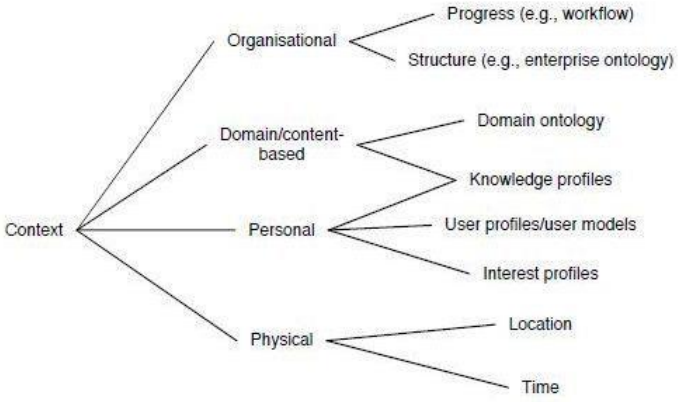

Figure 1. The topology of context (Topcu, 2011)

A detailed survey and analysis of context modeling approaches, techniques, context representations, and frameworks can be found, for example, in (Bettini et al, 2010).

The most recent proposal of the conceptual representation of context, designed specifically for current IoT systems, was issued by the IoT-A consortium (Nettsträter et al, 2012). The domain model of the IoT context, which captures the main concepts at a high level of abstraction, is presented in Figure 2. The *User - Physical Entity* interaction is the core relationship in this model, where both sides of the predicate are further expanded into generic classes of physical devices (blue boxes), software artifacts, services, and resources (green boxes) or human users (yellow box). The IoT-A model is intentionally quite simple and general, capturing all the most important concepts and relations for representing any context-relevant aspects that may be applied in IoT systems of all types. From this perspective, the IoT-A domain model provides a reference semantic architecture that can be reused and instantiated to obtain a custom representation of the context in ebbits, as it is described in the following section.

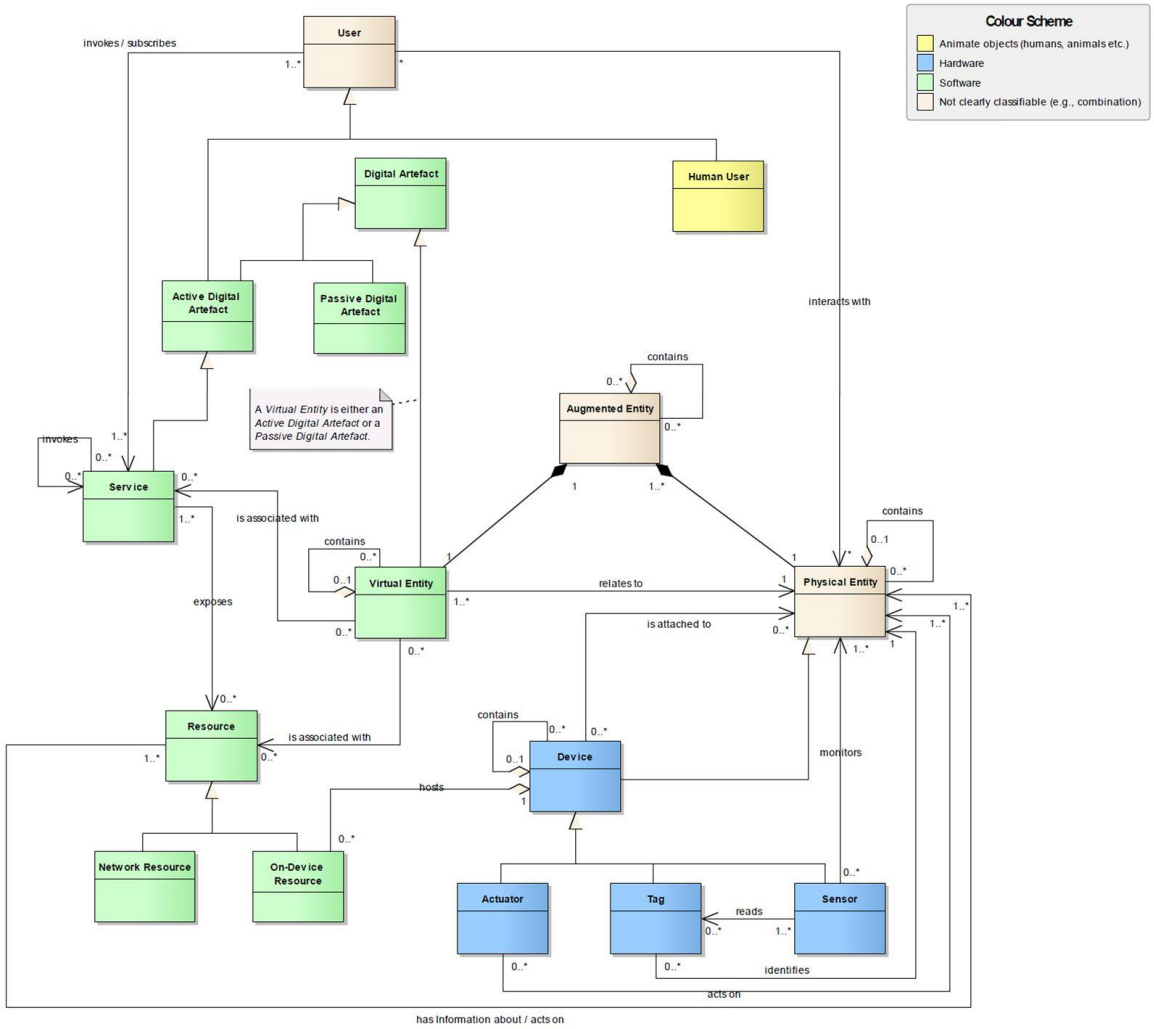

Figure 2. The IoT-A reference architecture - the IoT domain model

4.2 Semantic modeling for IoT in ebbits domain

The semantic modeling in ebbits extends the Internet of Things reference architecture proposed by the (IoT-A consortium 2012). We extend this model by introducing the context of the human users as people is aimed to be the key differentiator of ebbits.

As depicted in Figure 3, the reference context model above shows relationships that are often needed for developing IoT applications (showed in blue). The relation to the people (showed in green) is designed for supporting people in performing their work.

The following description explains how the main concepts depicted in the Figure 3 should be used in an application:

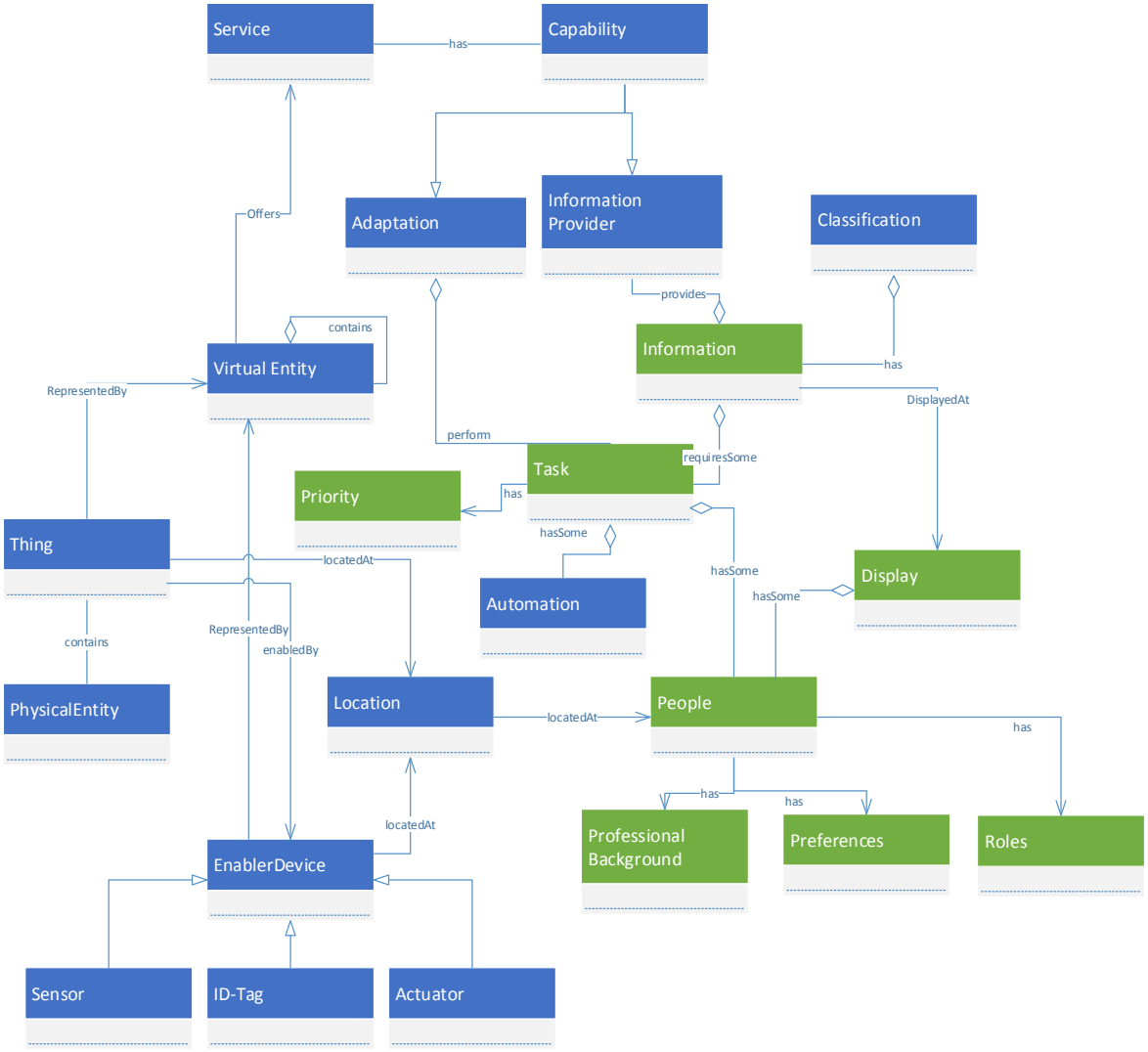

Figure 3. Reference context model for IoT

Thing

Thing in the IoT applications refers to a concept of smart objects which a user can interact with. Being part of the IoT, an entity must be addressable through some identification and provide at least a service to get its descriptions and other context parameter such as its location, services, and its quality parameters.

From the user perspective each thing is an integrated unit although it may be composed of several independent components that are explained in the following sections.

Physical Entity

Physical entities are tangible objects surrounding us. Physical entities can range from integrated electronic or a mechatronic devices which composed of software and hardware components offering services such as mobile phones, automation robots, embedded systems. Thing could also be a non-electronic device that must be represented by a composition of virtual entity or software objects and enabler devices such as sensor, actuators, and ID-tags.

Enabler Devices

Enabler devices are devices needed to enable physical entities to be part of the internet of things. The function of the enabler devices can be categorized to:

- Provide unique identifications.
- Provide the context of things.
- Provide means for adaptation.
- Provide means of communications.

Tags such as RFIDs and barcodes have been used to provide identification for products, while network addressing has been used to identify electronic devices with communication module integrated.

Sensors are used to provide the context parameter of things such as temperature and light sensors, GPS for location, and image processing. User input is also used to identify some parameters that cannot be reliably detected with sensors.

Actuators on the other hand allow adaptation to the context parameter of the things to be executed.

Virtual Entity

Virtual entity refers to software instances that maybe a representation of physical objects or it may also be completely virtual objects that do not represent any physical objects. In ebbits case, a complete virtual entity may represent a legacy system and business systems.

Virtual entity enables interactions to physical devices and software modules being expressed in software instructions / programming language which is encapsulated as services.

It also provides a standardized communication protocol that allows interoperability among the internet of things. For instance, in ebbits, LinkSmart proxies provide a web service communication to solve syntactic interoperability among heterogeneous protocols.

Services

Services represent functionalities that are provided by smart things such as retrieving its information, perform actuations. Services are accessible by sending instructions to its application programming interface (API). The services encapsulate the access to hardware and software components that form the smart "Thing". Nowadays there exist some efforts to use web technologies for standardizing the access to these services. E.g.: device profile for web service (DPWS), Constrained Application Protocol (CoAP). There are several proposals to standardize context information of web services e.g.: OWL-S, WSMO. However for ebbits use case, we try to find a balance between modelling and the usefulness of the context information. Based on the requirements, the capabilities of services are the only important context information for searching and discovering services.

People, background, and preferences

Ebbits aims at putting people in the center of the platform. People interact with internet of things in their work not only to retrieve information, but also to perform some adaptation to the environment. People also have some preferences that may influence the way they work, and therefore also affecting their ability to consume and perceive the information. The information shown to the user should be adapted according to their roles and tasks in the organization. This approach could help user to gain situational awareness without being overwhelmed by the amount of information.

The ability of the users to consume information depends on their professional background such as educations and experiences. Moreover they also have personal preferences how the information should be presented. Considering these parameters may help users understanding the information presented to them.

On the other hand, tailoring the presentation of the information according to the different context of the users may require a large amount of development efforts therefore the application developers must find the balance of customizing the applications to the benefit that the users would gain.

Tasks, Roles, and Priorities

Since the goal of ebbits is to establish internet of thing particularly for business environment, user tasks should be treated as the first class citizen that influences how the system works. Many systems adapt the presentation of information depending on the user roles. However these systems could be enhanced further by adapting the information based on the context of the tasks. This is particularly useful in a setting like in the manufacturing use cases where a massive amount of data come from the shop floor. The data is transformed into various reports that can be shown to the users depending on his tasks. E.g.: technician that is looking for an error need to see the detail of abnormal data from each devices, while a line supervisor may only be interested in the total throughput of the line.

Tasks may be executed by people as well as automated systems. When they use the same resources, conflict may happen and therefore prioritization of these tasks needs to be had.

Display and user location

The increase of post desktop devices with many form factors e.g.: phone, tablet, convertible or phablet open different possibilities for presenting the information in such a way that is optimal for the screen real estate. On a small mobile phones screen the user may prefer to have a simple list of updates, while in a bigger screen such as tablet the users may be able to see the more information.

User locations and which device the user is currently working with is an important factor that has not yet been addressed by many systems. Location may be used to infer several situations such as whether the users is accessing information with a public network that needs to be secured, or the users have the bandwidth to download all information. These considerations can be derived from the location of the users with the combination of other inputs in order to increase the certainty of the inference

4.3 Conclusion

In this section a context model reference that can be used by developers to develop smart IoT applications that consider the context of the users. There are several approaches for modeling user context depending on the focus of the application domain. Several approaches claimed to provide a standard ontology that can be reused for building applications for ubiquitous computing (Bettini et al, 2010). These approaches however have a lack of support work activities as envisioned in ebbits.

We have presented an extension to IoT-A model to support ebbits scenarios that involved interaction between IoT and the users in performing their work activities. This context model can be extended and combined with application specific domain model.

Developers may adjust the application behaviors according to the context of the users to achieve personalization, however modeling different behaviors may require a lot of efforts and increase the complexity of the system. The developers must be able to find the right balance between the detail of personalization and their benefits for the users as well as for the developers.

5 Ebbits Visual Context Modeling Tool

Ebbits aims at providing developers with Model Driven Development (MDD) tool that simplifies IoT application development by exploiting the Internet of Things model described in the section 4.2. The envisioned tool should allow novice developers to rapidly developed context monitoring application using ebbits.

To support this goal, we choose visual modeling tool that could define the relationships among sensors, actuators, and their semantics in terms of application domain objects. The tool will then generate the necessary software artifacts that represent the domain objects that can be access programmatically. This way, developers only need to deal with domain objects as they perceive the application domains without having to know the technology details of the sensors and actuators. The sensor and actuator drivers will be added and as plugins to the tool, which is going to be continuously maintained by the LinkSmart open source community.

MDD focuses to develop and to refine the model of a specific domain that provides user defined standard vocabulary that can be used in the development (Gronback, 2009). Using MDD for implementing a complex system could have the possibility to reduce time and cost of production(Bordin, Panunzio, & Puri, 2008). In MDD, a modeling language is used to create an abstract model of the system to achieve a better overview of the whole system.

There are several accepted standard modeling languages for building IT systems such as Unified Modeling Language (UML), Data Flow Diagram (DFD), Fundamental Modeling Concept (FMC) and System Modeling Language (SysML). SysML is a general-purpose language for system engineering applications. It supports defining the requirement, analysis, design, verification, and validation of a wide range of complex systems. These systems may include notations for describing hardware, software, information, processes, personnel, and facilities. It reuses a subset of UML 2 and provides additional extensions needed to address the requirements for system engineering (Object Management Group, 2010). FMC is a universal notation originating from existing notation standard. It clearly separates conceptual structures from the implementation structure to focus on human comprehensions of complex systems on a higher levels of abstraction compared to UML. FMC enables group of people to share a common understanding of a system's goal and structure by providing the concepts to create and visualize models without going in depth (Knöpfel, 2007).

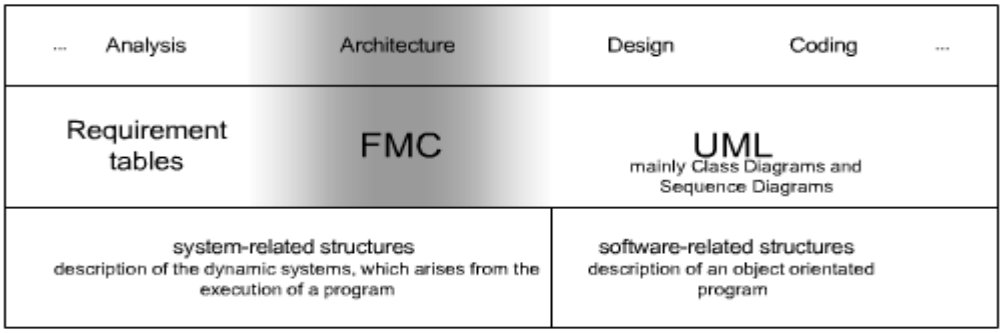

Figure 4. FMC fills the gap between UML and Requirements.

Alternatively, a Domain Specific Language (DSL) can be used in MDD. DSL is designed specifically for the intended tasks (Gronback, 2009). Gronback also claims that the cost of learning a complex language might be too costly and the effort of standardizing modeling language for general use cases will likely fails due to different requirements of the tasks, and the human nature to express their own creativity ("Not invented here syndrome").

After performing observation and interviews to our users, using a set of standard modeling language such as UML is too complex to support the intended tasks, we decided to implement a simplified DSL that consists a subset of UML and the needed extensions for modeling the association among devices and the virtual objects.

5.1 The Development Tool

We have gathered several developers and student assistant in Fraunhofer FIT to discuss the needed requirements for the context modeling tool. Based on the discussion with the users, we concluded that the users need to have a system that allows them to associate the sensors and actuators with the context parameters belong to the entities in the domain. They would like simply to deal with simple plain java objects to access the entities and their context parameters once the mapping between sensors and the objects has been defined. Moreover, they would like to be able to pre-process sensor data by applying fusion, filtering of the sensor data.

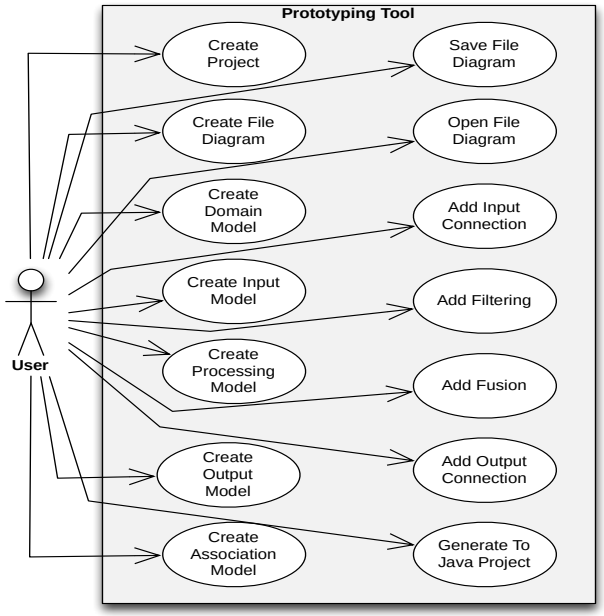

Figure 5 Use case of the prototyping tool

The following description explains the use case of the context modeling tool:

UC1. Create Input Model and Add Input Connection

The user is able to create an input model as an abstraction of the input connection e.g. connection to sensors, actuators, or other input software, or devices.

UC2. Create Processing Model and Add Filtering and Fusion

The user is able to create processing model as an abstraction of a processing filtering or fusion data.

UC3. Create Output Model and Add Output Connection

The user is able to create an output model as an abstraction of the output connection e.g. output connection to Graphical User Interface (GUI), console interface or database.

UC4. Create Association of Input, Processing, Domain and Output Model

The user is able to create an association from the input to processing, from processing to the domain model, and from the domain model to the output.

UC5. Generate diagram as Java Project

The user is able to generate diagram into a Java project that can be use as prototyping software of the system and extend to a full fledge of software.

UC6. Save and Open File Diagram

The user is able to save the prototyping diagram for later use or as an archive and open it again to make changes.

5.2 The workflow of the prototyping tool

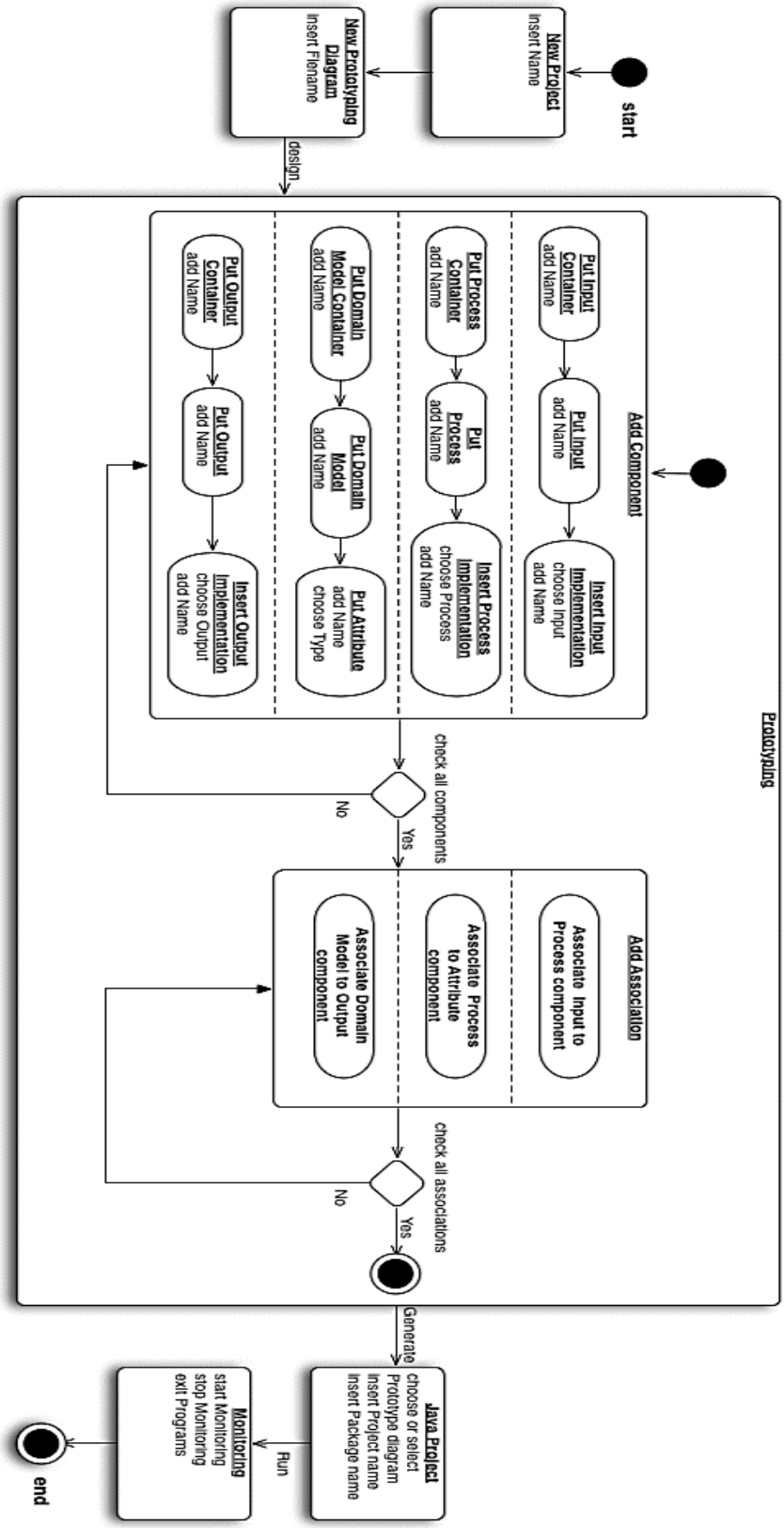

Figure 6. Workflow of the tool

The workflow of the system is started with creating a new project on the Eclipse development tool (Figure 6) where the user can enter the name of the project. Next, the user can create a new prototyping diagram inside that project and enter the filename of it. Then, the user may start designing the domain model of the context aware system. When designing the model, the user is able to add several components such as Containers for the Input, Process, Domain model, and the Output components as well as the corresponding components themselves. Each component container will contain specific component e.g. Input container component contains Input component. This container is design to avoid clutter of the diagram when modeling a large system.

The Input, Process, and Output components have several implementation components that the user can choose from e.g. Input component has an implementation of connection to the Arduino board and Plugwise, the Output component has an implementation of displaying the information to graphical user interface (GUI), Process component might have implementations such as moving filter, average, min, max.

The Domain model component does not have any implementation component, but it has the attribute component that defines what kind of data type and name that a Domain object will contain e.g. Domain model of room would have the name of attribute temperature, which type is double.

After the user adds all required components the user needs to associate the components as depicted in Figure 7. The association would be Input to Process component, Process to Attribute component, and Domain model to Output component. After all necessary association finished the user may start to generate the java code based on the diagram into prototyping of monitoring system. Then, the user can start monitoring of the system by selecting start menu, or stop the system by selecting the stop menu.

5.2.1 User Interface Design

Based on the use cases, a design of a context modeling tool has been created. The tool consists of three main parts: An editor view in the middle that allows developers to associate the entities in the domain with the context providers that deliver the context values. The Input box can be extended to include LinkSmart proxies e.g OPC, Arduino, sensors, and PLC. The data from the context providers in the input box as described in the Figure 7 must go through process layer where the data can be fused, aggregated, filtered before it is assigned to the context attribute of an entity. The developers are able to extend the tool by adding more of fusion algorithms. The process box may contain several processing boxes containing algorithms to fuse sensor streams. Each of the processing box is then associated with context property of an entity.

The tool should be able to generate Java source code create the necessary artifacts to start a development project (e.g.: Eclipse project). The source code generator should be extendible to generate different programming languages such as C, C++, Python, Objective C or Perl. However, in this period, in WP5, we will initially focus on generating a Java Project. The java project will contain all the necessary artifacts needed to start the development e.g.: Java source codes, libraries, and eclipse project configurations that are needed by eclipse.

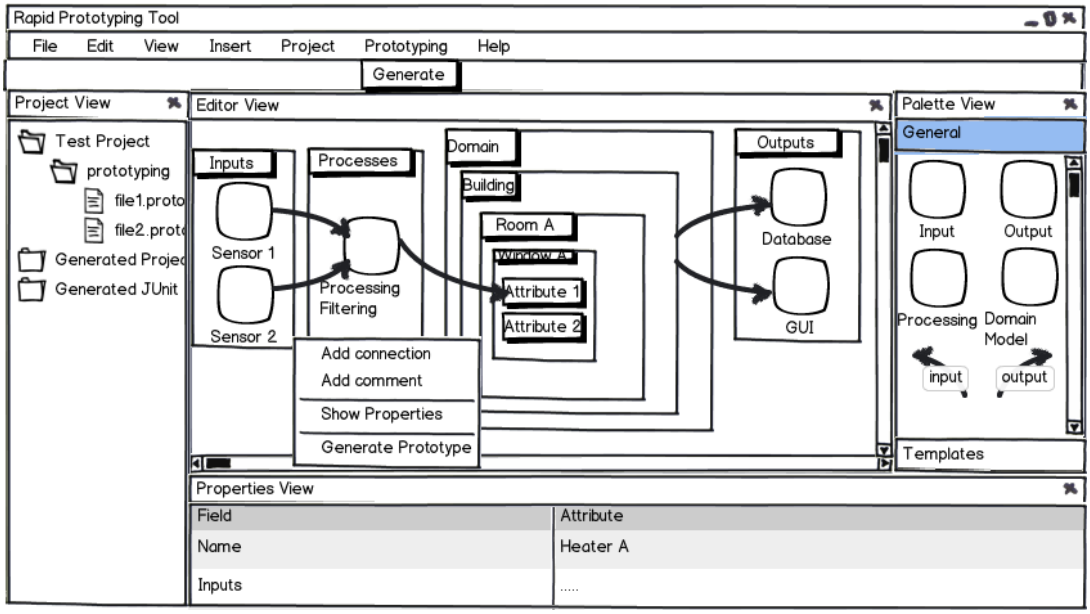

Figure 7 The mockup GUI of the prototyping tool

As shown in the mockup of GUI, the tool will have several views, which are Project View, Editor View, Palette View, and Properties View (Figure 7). It will have a button in the toolbar to generate the necessary artifacts such as the code of the real-time monitoring system. Project View can contain project of the user that can have many prototyping diagrams and the generated real-time monitoring system as a java project. Palette view contains several blocks and links of the model diagram that can be put in the Editor View, and it is arranged using tabular depending upon the type of the model. Editor View visualizes the model and the link of the block diagram that is put from the Palette View. In this view, the user can design their system using graphical programming or MDD according to their project requirements. In the Properties, View users can modify the properties of6 the blocks of the model diagram. Using this view the generated system will have real implementation of the system for the input, processing, and the output. Since the prototyping tool is intended as an Eclipse Plugin Project, it might look like as an Eclipse Development Tool.

5.3 Context Monitoring Architecture

The context monitoring system is meant to be generated by the tool designed in the previous section.

Requirements addressed:

- EBBITS-309 ebbits platform should have a publish-subscribe system
- EBBITS-223 The system provides access to aggregated/selected information through filters or fusion
- EBBITS-397 Prototyping tools for context modeling
- EBBITS-398 Java based object oriented context modeling
- EBBITS-328 Sensor fusion algorithms must be realized as a decoupled component.

Based on the developer workshops that was held at Fraunhofer FIT and the WP5 requirements, we have defined the use cases for monitoring the context of the virtual objects defined by the developers.

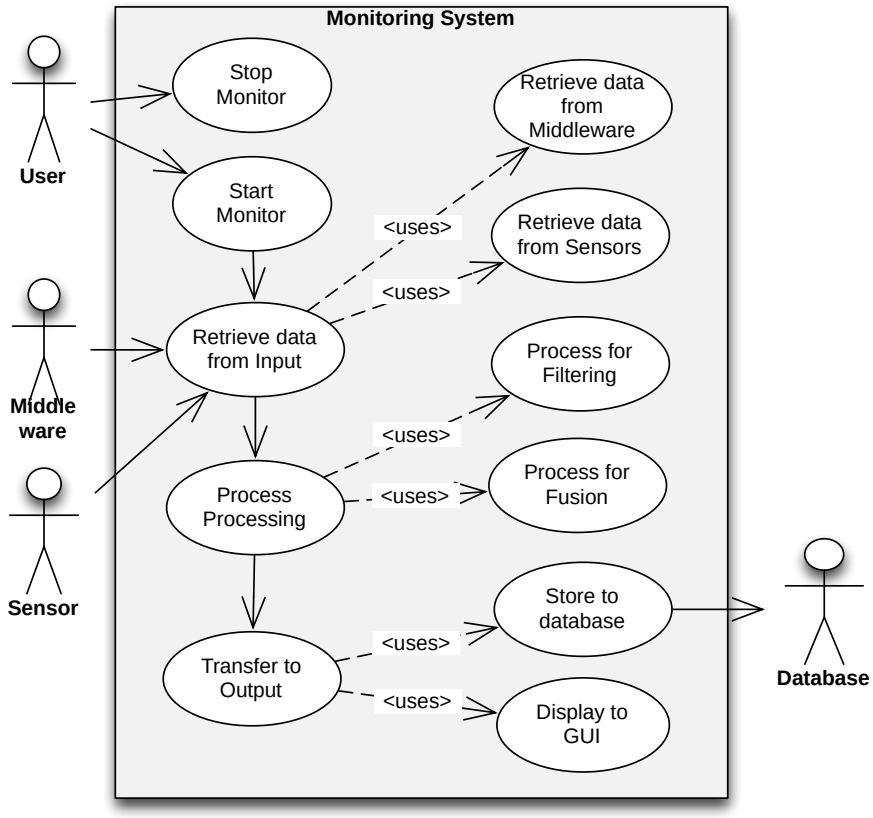

Figure 8 Use case of monitoring context of the virtual objects

UC1. Start Monitor

The user is able to start monitor of the system.

UC2. Stop Monitor

The user is able to stop monitor of the system.

UC3.1 Retrieve data from the sensor

The system is able to retrieve data input from the sensor such as retrieving data from sensors directly, using serial or communication port, or using middleware.

UC3.2 Retrieve data from middleware

The system is able to retrieve data input from the used of middleware such as LinkSmart or OPC[3].

UC4. Processing of input data

The system is able to apply pre-processing to the sensor data e.g.: filter input data according to minimum or maximum of specified value, fuse of sensor data to increase its confidence as well as inferring new information. This data processing will deal with a low level sensor filtering to reduce network traffic. A more intelligent sensor processing will be done by the work in WP7, specifically by the Event Processing Agent component.

UC. Transfer data to the output

The system is able to transfer data to the output e.g. store data to the database or display to GUI or Console Interface.

[3] http://www.opcfoundation.org/

5.3.1 Architecture block of the context monitoring system.

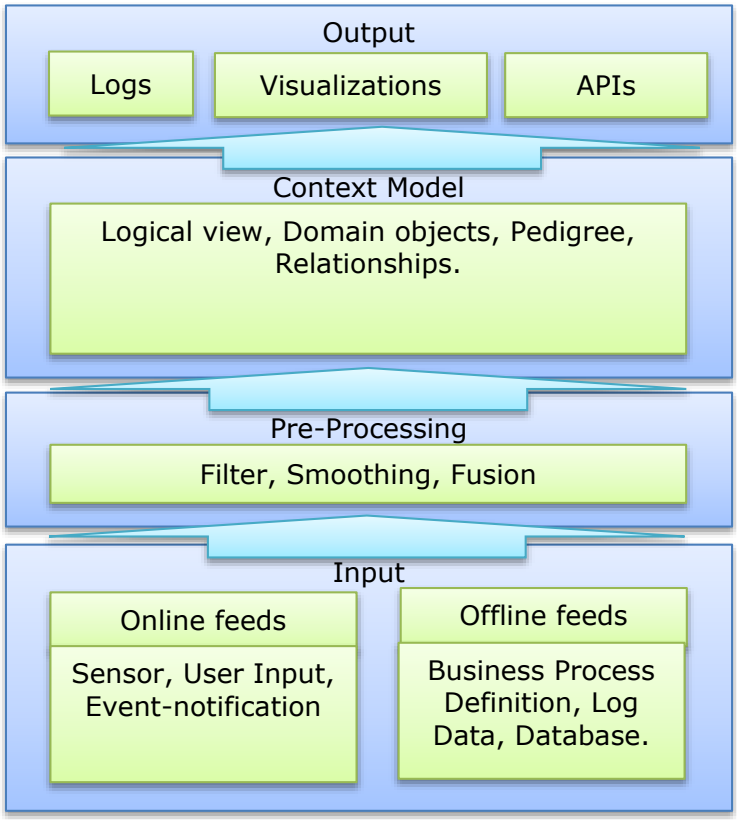

Figure 9 Software architecture for monitoring context of IoT

Our design approach keeps the architecture simple and making it familiar to the developers in order to reduce the learning curve needed by novice developers. The general architecture is divided into four components of input, processing, context model, and output component (depicted in Figure 9). In the input component, there are two categories of inputs that we could use such as on-line feeds and offline feeds. On-line feeds may contain hardware drivers such as sensors, actuators and PLCs (Programmable Logic Controller) as well as software proxies such as web-services that provide online data streams. In the offline feeds the input can contain artifacts that access historical data such as databases, log files, business process definition, and formatted files for instance CSV (Comma Separated Values). This component can be abstracted using the existing LinkSmart middleware drivers.

In the Pre-Processing component, several basic processing modules will be available as plugins. The basic plugins that will be provided includes data filtering, averaging, high-pass / low pass filter, or even filter for interpolating data (e.g.: Kalman Filter). Fusion module can be used e.g. to fuse data from different sensors in order to augment the confidence of the data as well as to derive information that is not possible to be detected by a type of sensor.

The context model component contains the virtual objects and their associations to each other as well as the associations to the atomic devices such as sensors and actuators based on the mapping defined by the developers. Once the mapping is done using the provided tool, this model will be generated in a native programming language.

The virtual objects in the context model may have their own properties such as position, temperature, intensity of the light, energy consumption. The data stream from the input component such as sensors will automatically update the context values of the entities after it has gone through a pre-processing modules. This simplifies further the application development since from this point on, the developers only need to deal with the virtual objects through plain java objects (POJO). For example, when the temperature properties

of the domain object, or room is mapped to sensor temperature, the properties will automatically be updated accordingly. The developers can simply access the properties of those domain objects without worrying where the value of the properties comes from.

The output component may contains components that visualize the real time values of the virtual objects, components that log the values into e.g. database, and API components that provides access to the virtual objects for the external applications through e.g. a web service.

The flow of the data initially comes from the input component, which is then processed in the processing component. After the data that come from the input gets processed, it is stored in the properties of the domain objects. The output component will be responsible to monitor the data for the properties of the domain objects and display them for real-time monitoring applications or dump them to a file or a database.

The processing component can handle the data from several input components. The output component can also handle many attributes of the domain model. The domain objects can contain other domain objects e.g. a room is inside a building, and a light is inside the room.

5.3.2 Logical View of IoT Context Monitoring

Figure 10 shows the class diagram of the system that monitors the context of virtual objects. The class diagram is a meta-model that will be used for defining the structure of the DSL to specify the domain model. The flow of the system will be the same as the concept from input into processing, from processing into the attribute of the domain model and from attribute of the domain model into the output component.

The classes are abstractions that specify the common structure and behaviors of a set of objects (Bruegge & Dutoit, 2004). It is divided into five layers and contains several classes. The layers and classes are described below:

First Layer (red rectangle)

This layer concerns with the user interface such as GUI for controlling and monitoring system. It contains only one class, which is UIController.

UIController: It has several methods such as createControl for initializing elements of the UI controller, startMonitor for starting the monitoring, and stopMonitor for stopping the monitoring process.

Second Layer (light blue rectangle)

This layer concerns as a main module of the monitoring system. It also contains only one class, which is the Monitoring class.

Monitoring: It has methods, which are the "setup" method for setting up the entire object's creation and objects mapping, "startMonitor" for starting the monitoring process, and "stopMonitor" for stopping the monitoring process.

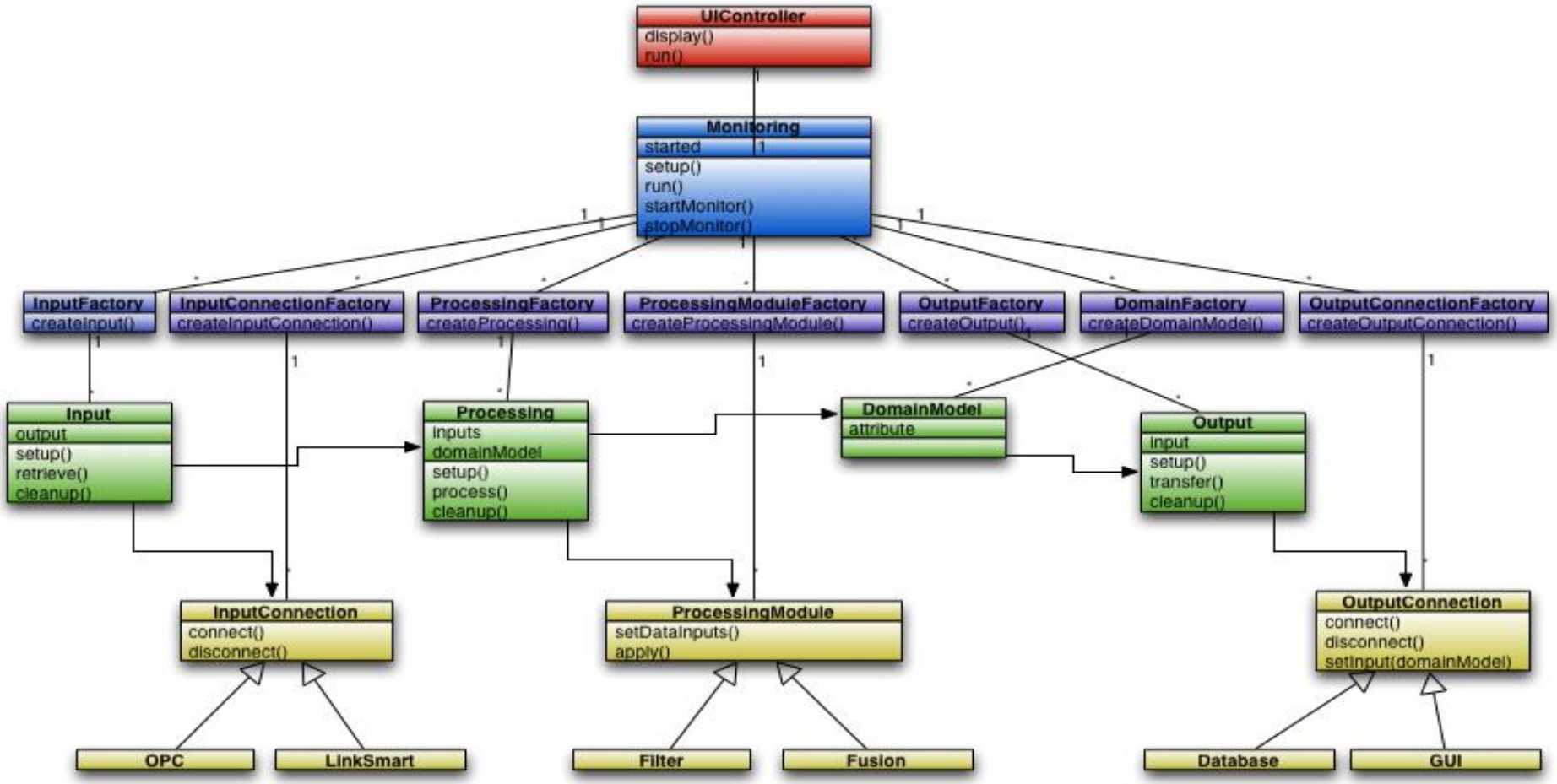

Figure 10 Class Diagram of monitoring system

Third Layer (dark blue rectangles)

This layer deals with creating all objects from the classes in the green layer. The classes are:

- InputFactory: deals with creating all the objects for the Input class.
- InputConnectionFactory: deals with creating all the objects for the InputConnection class.
- ProcessingFactory: deals with creating all the objects for the Processing class.
- ProcessingModuleFactory: deals with creating all the objects for the ProcessingModule class.
- DomainFactory: deals with the creation all the objects for domainModel class.
- OutputFactory: deals with the creation all the objects for output class.
- OutputConnectionFactory: deals with the creation all the objects for outputConnection.

Fourth Layer (green rectangles)

This layer concerns all the classes that are related with all components of the logical view of the architecture and InputConnection class. The classes are:

Input: This class concern with the input data from InputConnection class to the Processing class. It is designed using an MCV pattern (Model-View-Controller) and designed as an observer in the MVC pattern that will get updated with the data from observable or InputConnection class or model in MVC pattern. The controller contains a middleware component or native driver of a particular input.

Processing: this class is designed using a separated thread for every Processing task, since each of the Processing task may use a Processing module such as filtering and fusion modules that can take some time to process the data. This way, long processes do not cause a bottleneck in the system. Each of the Processing task would have a task that execute a sequence of Processing modules. After the data from the input module is processed, the Processing class will update the attribute or property of the objects according to the relationships among objects and sensors assigned before in the Monitoring class.

DomainModel: This class represents the domain objects in the application domain and their relationships. Each DomainModel object can have several attributes. This class stores the data that come from the Input class after it is pre-processed in the Processing class.

Output: This class deals with the output of the system. Each output is designed as a thread. This class will monitor the objects in the DomainModel and then pass the data into any instance of the OutputConnection class such as GUI.

Fifth Layer (yellow rectangles)

This layer concerns with the implementation that is closer to the particular middleware or specific function. This Layer is an extension for all components in the fourth layer. The classes are:

InputConnection: This class extends the input component for any implementation of a specific middleware or a driver of the input source. The developers can extend this class to use for a different kind of devices. It is designed as a model in MCV pattern that will get an update of the data from the controller, which is the middleware or the driver of a specific input source. Using the InputConnection class, developers can extend the system to include a particular input source that can be incorporated using a middleware or driver such as OPC, LinkSmart, communication port (OS driver.

ProcessingModule: This class extends the processing component and can be designed for filtering or fusion processing. The user can implement different filtering and fusion processing to according to the functionality of the system. The developers only need to implement processing interface and "apply" method according to their own need. The Processing module will process all data from the input, which are passed by the Processing class.

OutputConnection: This class can be extended to a specific implementation of the output such as a report interface or a database.

5.3.3 Sequence Diagram

Sequenced diagram is used to describe the dynamic activities of the system and visualize the communication between the objects (Bruegge & Dutoit, 2004) of the virtual objects monitoring system. There are two possibilities of having the data into a system. First is by pulling the data every interval of time to the sensors (Figure 11). The second technique is the push, for which the sensors itself can send the data to the system using an event-driven paradigm (Figure 12).

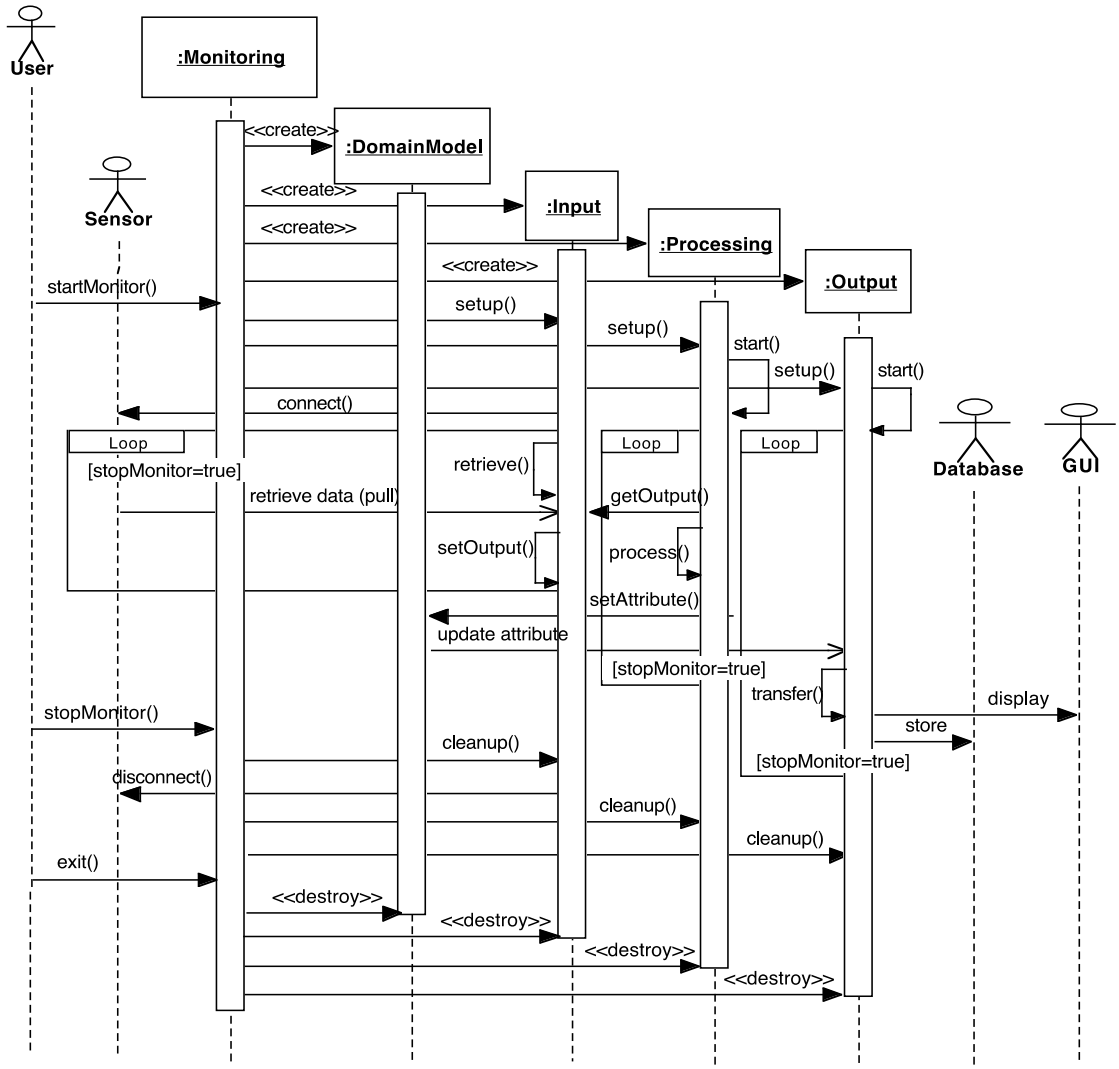

Figure 11 the pull version of the sequence diagram

First when the user starts the system, the Monitor object creates the necessary objects defined in the Domain Model, Input, Processing, and Output objects. After it has created the objects, the system starts monitoring the context of the virtual objects and sends the message to the Monitor object. Then it sends out the setup message which initializes the system. For example, the Input object will setup the connection to the specified. For the pull version creates a loop that retrieves the data from the sensor according to a specified interval.

In the meantime, The Processing object runs a loop to get the new data from the Input according to the specified interval. The Processing object calls a user-defined method that is responsible for doing the pre-processing such as filtering or sensor fusion.

The Processing object has been designed with the pull version that pulls the sensor values every interval of time. Since the processing can be defined by the developers, there is no way of knowing how long the process will take. For this reason, separate thread should be used for each processing module and synchronization of the threads should be taken care of. When the Processing object and the Output object are created together and running in separated thread. When it has finished processing the input data, it will set the attributes of the virtual objects with the pre-processed data. The Output object will parse the pre-processed data by accessing the virtual objects. The Output objects observe the data belongs to the attribute of the virtual objects in the domain model and then transfers the data to the specified output such as a database or GUI.

The monitoring stops when the user sends a message to stop it. The Monitoring object will subsequently send a "cleanup" message to all the objects that have already initialized, next stop the processing threads, and disconnect from the sensor, later it will destroy all the objects when the user exits the system.

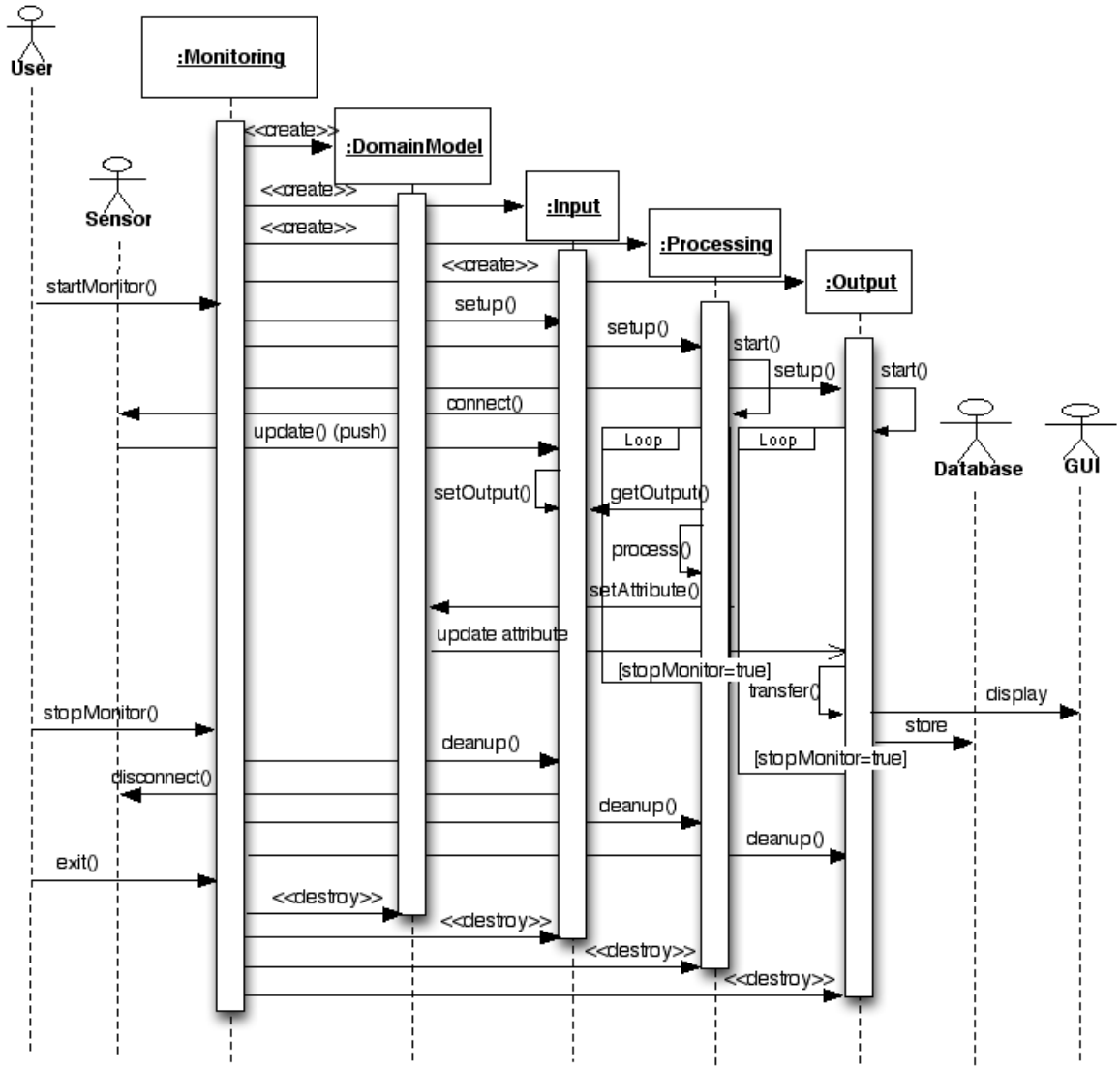

Figure 12 the push version of the sequence diagram

The push version follows a publish-subscribe pattern. Whenever the data is available, the Input object will push the data from the sensors into the processing, then to the virtual objects, and then the output component.

5.4 Conclusion

This section presents the architecture of the visual context modeling tool and the architecture of the code that should be generated to monitor the context of the virtual objects defined by the users. The implementation of this architecture will be reported in the next prototype deliverable.

References

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D. (2008). A Survey of Context Modelling and Reasoning Techniques. Journal of Pervasive and Mobile Computing, Volume 6 Issue 2, Elsevier, April 2010, pp. 161-180.

Bordin, M., Panunzio, M., & Puri, S. (2008). Rapid Model-Driven Prototyping and Verification of High-Integrity Real-Time Systems, 491-492.

Bruegge, B., & Dutoit, A. H. (2004). Object-Oriented Software Engineering - Using UML, Patterns and Java.

ebbits consortium. (2011). D2.1 Scenarios for usage of the ebbits platform.

Gronback, R. C. (2009). Eclipse Modeling Project A Domain-Specific Language (DSL) Toolkit.

IoT-A Consortium. (2012). D1.4 Converged architectural reference model for the IoT v2.0

Knöpfel, A. (2007). FMC Quick Introduction.

Nettsträter, A: Internet of Things - Architecture (IoT-A, 257521) (2012). Deliverable D1.3 - Updated reference model for IoT v1.5. IoT-A consortium, 2012.

Object Management Group. (2010). OMG Systems Modeling Language (OMG SysML TM).

Poslad, S.: Ubiquitous Computing. (2009). Smart Devices, Environments and Interactions, 1st ed. John Wiley & Sons, Mar. 2009.

Satyanarayanan, M. (2001): Pervasive computing: vision and challenges. Personal Communications, IEEE, vol. 8, no. 4, pp. 10 -17, Aug 2001.

Topcu, F. (2011). Context Modeling and Reasoning Techniques, SNET Seminar in the ST, 2011.