NTNU
Norwegian University of
Science and Technology

Modeling of mobile information systems

Espen Strømjordet

Master of Science in Informatics
Submission date: May 2017
Supervisor: John Krogstie, IDI

Norwegian University of Science and Technology
Department of Computer Science

ABSTRACT

Abstract

Conceptual modeling languages are used for among other things sensemaking and analysis of information systems. In the field of mobile information systems, there is a lack of modeling notations that cover the specific components which are common in this domain. This thesis explores creating a conceptual modeling notation suited for modeling the main components of interest for mobile systems. Particularly the Internet of Things (IoT) and sensor networks have been covered in the notation.

The approach used in this thesis has been a theoretical review of existing modeling notations to use as a base notation, which were further extended with constructs specifically targeting the mobile domain. Testing was conducted on an actual case to evaluate and improve the relevancy and usefulness of the notation.The result is a notation that is capable of representing a variety of IoT and sensor-related objects while maintaining best practices for visual modeling notations. The notation could readily be extended in the future to support other types of systems as well. In practice it still needs to be further tested and applied to other real cases, in order to accurately determine the domain appropriateness of the notation as well as whether it is comprehensible and useful for organizational needs.

Sammendrag

Konseptuelle modelleringsspråk brukes til blant annet å gi mening til og analyse av informasjonssystemer. I domenet mobile informasjonssystemer er det en mangel på modelleringsnotasjoner som er ekspressivt sterke nok til å dekke de spesifikke komponentene som er typiske i dette domenet. Denne rapporten dokumenterer et forsøk på å skape en konseptuell modelleringsnotasjon egnet til å modellere hovedkomponentene av interesse innen mobile systemer. Spesielt områdene tingenes internett (IoT) og sensornettverk har blitt dekket i notasjonen.

Tilnærmingen i denne rapporten har vært en teoretisk gjennomgang av eksisterende modelleringsnotasjoner som dannet grunnlaget for en ny notasjon, og deretter har blitt videre utvidet med objekter spesielt rettet mot det mobile domenet. Testing ble gjennomført ved å modellere et eksisterende IoT-relatert prosjekt for å evaluere og forbedre relevansen og nytten av notasjonen. Resultatet er en notasjon som kan representere en rekke IoT- og sensorrelaterte objekter, og samtidig opprettholder god praksis for visuelle modelleringsnotasjoner. Notasjonen har potensial til å bli utvidet i fremtiden for å støtte andre typer systemer enn det som har blitt fokusert på her. I praksis trenger notasjonen fortsatt testing og utprøving på andre eksisterende systemer, for å vurdere hvorvidt notasjonen dekker domenet på en god måte og blir oppfattet som forståelig og nyttig for organisatoriske behov.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Prof. John Krogstie for the advice and feedback throughout the duration of this project, as well as providing useful literature and assisting with the evaluation of the developed notation. I also want to express gratitude to Dr. Dirk Ahlers for his useful feedback during the evaluation of this project.

Contents

CHAPTER 1 INTRODUCTION

Conceptual modeling languages are an integral part of systems engineering. They are used for a variety of tasks such as requirements engineering, sensemaking for internal and external stakeholders, or for improving understanding about the system itself. A variety of modeling notations exist for different usage areas, and it is important to use a notation that is suitable to represent the most relevant parts of the system. General modeling notations are flexible in their ability to model many different systems, but specialized notations also exist, usually being more effective than general languages at creating models in their target domain [1]. This introductory chapter introduces the modeling problem to be solved in this thesis and gives an outline for the rest of the report.

1.1 Problem description

The Internet of Things (IoT) have become an important part of the Internet infrastructure, with "things" such as sensors and actuators that generate data which can be combined and processed into valuable information. The thesis presented here is an attempt at creating a conceptual modeling notation for the field of mobile information systems, focusing on representing unique aspects of these systems that other notations are insufficiently able to represent. This project primarily focuses on the mobile domains of IoT and wireless sensor networks (WSN), and applying the notation to a real case. Some key concepts for the notation are: Sensors, actuators, locations, data sources, data transformations, and data processing.

The metamodeling effort is concentrated on the analytical aspect of system modeling. It is intended to be used for sensemaking and communication rather than automatic execution. In a distributed mobile environment, many actors and many data sources exist. The main purpose of the developed notation is to support creating an analytical overview of how the multitude of data sources and actors interact, how data is transformed in the various steps, as well as providing high level overviews of these systems. The forms of modeling which have been focused on are data flow modeling and process modeling. The developed notation uses parts of various existing notations. However it is not merely an extension of pre-existing notations, but rather a standalone original notation that borrows elements from other standardized notations.

A case study has been used in this thesis as an example case that the notation should be able to represent. The case in question is the Carbon Track and Trace (CTT) project[2][3], which is related to Smart Cities and tracking the emission of greenhouse gases (GHG) in big cities. This is a topic that is also related to IoT and is therefore a relevant case for this project. The CTT case is described in more detail in Chapter 4.

Mobile information systems have unique properties compared to traditional systems. These include such factors as geographically distributed components, specialized devices, and complex interactions between many actors. In order to visualize systems like this, a specialized modeling notation could be desirable to ensure that domain knowledge is captured while keeping the complexity of the models reasonably low and understandable.

In order to understand the work in this thesis and the developed notation, it is an advantage if the reader is familiar with conceptual modeling of information systems, in particular the notations of Business Process Model and Notation (BPMN) and Data Flow Diagrams (DFD) which are central to the work in this thesis.

1.2 Structure of thesis

In this chapter, a brief introduction to the problem at hand has been given. The fields of IoT and sensor networks have been introduced, as well as the main goals and expectations for this project.

Chapter 2 features a review of existing research and literature that is considered relevant for the case, and current state of the art efforts which have been inspirations for this project. The chapter also provides some background discussion about the modeling notations that are featured in this thesis, to make the following chapters more understandable.

Chapter 3 describes the methods used for conducting the research and implementing the modeling notation. It goes more into detail regarding the goals of the project and the feasibility of achieving them. The steps involved in the creation of the notation are elaborated upon and justified with regards to the project goals.

Chapter 4 goes further into detail about the case which is used as a baseline for this project. It elaborates on the artifacts that make up the CTT systems as these make up the base for the modeling notation. This chapter also discusses how CTT relates to the domains of IoT and mobile systems.

Chapter 5 provides an in-depth breakdown of the implemented notation, covering details and origins of the modeling constructs as well as justifications for the choices made during the modeling process. This chapter also serves as a reference for modelers using the notation, as it contains the details of usage and properties of all the concepts found in it.

Chapter 6 evaluates the developed notation by using empirical evaluation of the testing results. Notable quality frameworks from existing literature are also used, to give a more analytical evaluation of the notation. Any discovered advantages and limitations of the framework are expanded upon in this chapter based on the evaluation results.

Chapter 7 discusses the project as a whole, how to interpret the results as well as threats to the validity of the results. This chapter also puts the project in perspective, as to how it contributes to the field of conceptual modeling and how current practices and the research approach have impacted the results.

Chapter 8 concludes the thesis with a brief reflection on the project including possibilities for future work. The main results and findings are summarized here.

CHAPTER 2
RELATED WORK

This chapter discusses existing work that has influenced the notation and modeling effort presented here, which includes reviewing pre-existing modeling notations that were found to be suitable for modeling aspects of mobile systems.

2.1 Existing projects

This section reviews projects related to IoT and conceptual modeling. Some of these projects may not be directly related to the modeling task conducted in this thesis, but could be relevant in advancing the scope of the modeling notation beyond creating conceptual models.

One qualitative study investigated how people understand visual models from a cognitive point of view [4], using the BPMN notation. This study highlighted in particular the importance of the participant's existing knowledge about notations and domains. A participant's knowledge of the modeling language or the domain being modeled seemed to be a central factor for the difficulty the participant would have with the language. As BPMN is a general language used to model many different domains, the domain knowledge of BPMN models is separated from the knowledge of BPMN itself. However in a domain specific notation like the one described in this thesis, domain knowledge will naturally have an impact on both understanding the models and understanding the notation itself. Therefore, users who have expert knowledge in the IoT domain would be expected to have fewer issues with a domain-specific language targeted at the IoT domain (assuming that the modeling notation is indeed an accurate representation of the domain). A different study compared participants with high and low domain knowledge of information systems [5]. The study found that application domain knowledge mainly has a large impact on problem solving tasks that involve conceptual schemas, while general knowledge of information systems impacts all types of tasks involving schema understanding.

In [6], a modeling environment was created specifically for IoT services and ontologies. In this environment it is possible to make formal descriptions and visualizations of IoT ontologies, which shows object instances and the communication between them. It is also possible to model sensors and actuators and query these objects for information. Another IoT project of note is the OpenIoT project[7]. This project provides a service framework with a variety of tools for IoT-based systems such as middleware for sensor networks, semantic models of the systems, and cloud-based services. IoT systems have significant challenges due to the size and geographical dispersity of these systems. In [8], an architectural approach is presented which details the composition and orchestration of large IoT systems. Another project which is relevant for complex IoT systems is the data MULE project [9]. This project analyzed the architecture of sensor networks and how to improve the architecture for saving power and reducing latency in networks of sensors. Architectural guidelines like this could be implemented in conceptual modeling notations as well, since the conceptual models are abstractions of the real architecture employed by the systems being modeled. Thus, if creating models of future systems, following efficient architectural guidelines could impact the cost and performance of actual sensor networks as well.

A modeling language called ThingML[10] was developed by the research organization SINTEF with the purpose of providing model-driven tools used to develop embedded systems. The focus on embedded systems can be tied in to the mobile IoT systems worked on in this thesis, as these systems often feature embedded components. This thesis primarily considers conceptual modeling rather than system generation which ThingML provides. Despite this, many of the properties of ThingML are transferable to conceptual models and some of the ideas presented in ThingML may apply for this type of modeling as well. ThingML takes into consideration that embedded devices typically have many limitations such as CPU, memory and battery power. These limitations could also be a relevant consideration for a conceptual modeling notation which is the case here. The formalization presented in ThingML may be a venue for extension of this notation in the future, if system generation is desirable.

2.2 Existing notations

This section briefly discusses the existing notations used in this project and potentially useful extensions of BPMN.

2.2.1 Data flow diagrams

DFD is a high level conceptual modeling languages mainly used for showing the movement of data between entities. The main purpose of this notation is to create architectural overviews of systems with a low level of detail. DFD makes it simple to visualize how data moves throughout a system, how data is processed and where it is stored. This makes DFD useful for modeling systems where movement of data is common.

Processes may be used to show how data is produced and consumed, but not on the same amount of detail as a dedicated process diagram. Traditional DFD imposes very few restrictions and rules upon modeling. Using the ideas from DFD, the main points of interest are the flow of data between physical entities, as well as the processes that manipulate this data. In an IoT setting, DFD entities represent IoT objects such as sensors and gateways, showing how data is moved, aggregated, and processed throughout the system. Since traditional DFD does not provide for visualizations of details, rules and restrictions needed to be imposed on the notation to achieve better accuracy and formalization of the models.

2.2.2 BPMN

BPMN allows for detailed modeling of business processes. A process is contained in a swim lane, and there can be multiple swim lanes in a swim lane diagram. With BPMN models it is possible to illustrate the inner workings of the data exchanges, manipulations, or any other part of the system. This leads to possibilities of better insight and analysis in the model. Currently BPMN is a main standard for process modeling, and is consequently widely used and understood by business stakeholders.

BPMN is a notation consisting of many concepts. There are basic elements which are extended with specialized types for special cases. Some objects are commonly found in most diagrams, such as activities and events, while other objects are considerably less frequently used. The entire scope of the BPMN 2.0 notation was excessive for the requirements of the new notation, so a basic subset of elements was considered sufficient. This subset is described in Chapter 5 and discussed further in Chapter 6. BPMN is strictly used for business process modeling, and since the new notation included structural and data modeling as well, integration was necessary between BPMN elements and non-BPMN elements.

2.2.3 Extensions

There exist several attempts to extend BPMN to include aspects of other domains, of which some notable ones regarding mobile systems include:

- An extension to BPMN exists that enables BPMN to support choreographies between one-to-many and many-to-many interactions, among other things [11]. This extension introduces the concept of *sets*, which are multiplicities of objects. The extensions makes it possible to model interactions and referencing in BPMN on a level of detail which is not possible in standard BPMN.
- BPMN4WSN extends BPMN to support WSN [12]. This notation extends traditional BPMN with the concepts of WSN task, WSN pool and performance annotations on tasks. The extension simplifies business processes involving sensors and networks, and also allows for a higher level of detail in modeling these domain-specific concepts.
- A 2013 project attempted to add formal descriptions of IoT devices and services to standard process modeling [13]. This was an early approach toward mapping IoT resources to formal models and being able to generate systems that included these resources. While this approach proposes stricter and more formalized rules than what is required for a conceptual notation, it could be of interest if formalization of the notation is going to be relevant in future work.

Not many attempts at making extensive conceptual modeling notations specifically for aspects of IoT or mobile networks have been found at the time of writing this thesis. Therefore, some of the existing projects and studies discussed in this chapter were used as starting points and/or guidelines for this notation. The notation brings together concepts from several of these sources and adds some new concepts with the goal of creating a notation that is both extensive and understandable. Chapter 5 provides a more in-depth description of every object in the notation and their sources.

CHAPTER 3 RESEARCH METHOD

This chapter discusses the main goals of the research and details the approach used to meet these goals, including the reasons for the methods that were used.

3.1 Research goals

The goals of this study coincide with the general goals of any conceptual modeling notation. Some general metrics of quality for a conceptual modeling language are described in the SEQUAL framework described in (Krogstie, 2012) [14], which include that the language should ideally be (in general terms):

- Suited to model the parts of a mobile information system that are deemed relevant by those involved in the modeling efforts.
- Understandable by modelers and viewers of the model based on empirical principles for good visual design.
- Having a strong correlation between the participants' perception of reality and the constructs used in the modeling notation.
- Lending itself to representing all the knowledge held by the modelers.
- Consistent with current organizational standards to facilitate adoption of the notation.

Naturally, no modeling language is without flaws or compromise. For example, increasing the expressive power of a notation tends to reduce its understandability. The notation should therefore strive to reach an acceptable balance of meeting the various goals. The extent to which the notation was able to meet these goals has been determined through the testing and evaluations detailed in Chapter 6. The notation's primary function was to be a prototype for experimenting with this type of domain specific modeling. That is also why a test case was used in order to evaluate the applicability of the notation in a realistic setting.

3.2 Research approach

The research methodology used for developing the modeling notation is twofold, consisting of a theoretical literature study followed by empirical and analytical testing.

Initially a theoretical approach was used to create an initial version of the notation. This approach involved studying previous projects and articles in order to get a sense of which concepts were most relevant to be able to represent conceptually. These previous works were discussed in Chapter 2. This primary research was focused on the subject of IoT and sensor networks. Relevant concepts regarding IoT were chosen to be used as conceptual objects for the modeling notation. Anything that was considered relevant for the purposes of this modeling language were candidates to be used in the metamodel. Based on the primary study, an initial conceptual metamodel was built in the Metis platform, developed by Troux[15]. This tool is not widely used in practice, however it facilitates rapid creation of new modeling constructs with usage rules, and many templates of popular modeling languages exist within it such as BPMN and Unified Modeling Language (UML). The details on how to access the metamodel within Metis can be found in Appendix B.

For the second part of the research, the initial metamodel was tested on the CTT project, which had previously been decided on as a target case for the metamodel. This was done by preparing example models and consulting with one of the researchers on the CTT project to give feedback on the models. This provided an evaluation of the model at that point and showed how the model needed to be further improved to fit with the CTT case, which led to further additions and refinements of the model. The benefit of focusing on one relevant case was to get a starting point to establish the notation as relevant and useful for modeling this domain. Alternatively multiple test cases could be used, or the initial notation could have been made without a test case. However, with multiple test cases there might have been a conflict between the interests of each case, leading to a notation that could not sufficiently model any one case. Conversely if the notation had been made without a test case, evaluating its usefulness on an empirical level would have been difficult.

CHAPTER 4

CARBON TRACK AND TRACE

This chapter contains a case review of the CTT project, which is a project related to the fields of mobile systems and IoT, and was therefore desirable to be able to make models of. The contents of this review are mainly based on the various deliverables available for the CTT project [2][3].

The CTT project attempts to provide cities with tools to improve the monitoring and reduction of GHG emission in the atmosphere. This is done by placing sensors throughout various cities in the world and collecting real-time data of emission. The sensors involved are typically low-cost alternatives with limitations such as range of detection, price, and types of gases measured. Sensors are connected to sensing units, which are responsible for sending the sensor data through the network to cloud storage. Each sensing unit may have multiple sensors connected to it, and the sensing units are identified by hardware ID or MAC address. Figure 4.1 shows one such sensing unit, installed with attached sensors and powered by a solar panel. A variety of sensor data types are captured and stored, including GHG emissions, air quality data and temperature data.

The sensor data from the various sensors around the city is sent through scalable gateways and collected in a cloud platform. The gateways are locally installed devices that receive sensor data encoded in the LoRaWAN protocol, which they then forward over a TCP/IP wireless network to the analytical backend. The gateways used are Kerlink gateways as shown in Figure 4.2.

The collected data sets are analyzed on an analytical backend. These systems are designed to be able to integrate with existing IoT systems in the cases where cities already employ such backbone systems. The backend is managed by The things network (TTN). All nodes and sensors are part of a pool where every device in the pool has a unique ID to identify it. TTN manages this pool and applications can get the data from individual devices by accessing them through TTN, identified by their ID in the pool. Figure 4.3 illustrates this interaction.

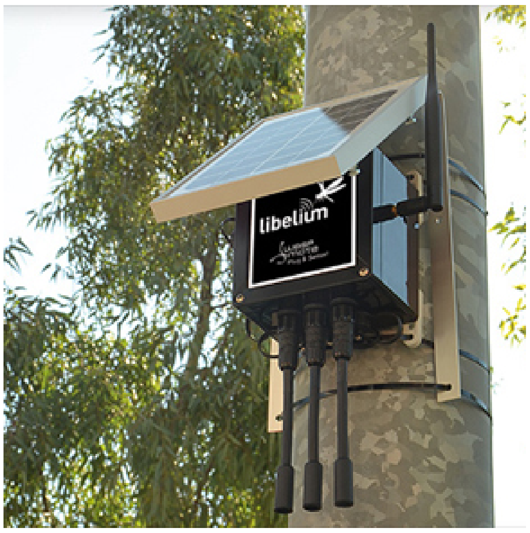


Figure 4.1: A libelium sensing unit with attached sensors (Source: `http://www.libelium.com/`).

Figure 4.2: A Kerlink gateway used in the CTT project (Source `https://www.thethingsnetwork.org/`).

The state of the sensor network is monitored by the DataPort tool, which creates visualizations of all the connected networks in the pool and their signals. The captured sensor data is eventually stored on a MonetDB database before being sent to an analytics engine. The analytics engine uses a combination of the captured data as well as external historical data to create visual representations for end users. It is one of the main goals of CTT to provide detailed emission data to end users such as decision makers and general citizens. These visualizations can be made for a variety of purposes, such as illustrating the correlation between traffic and air quality, or showing the variance in GHG emission over a period of time.

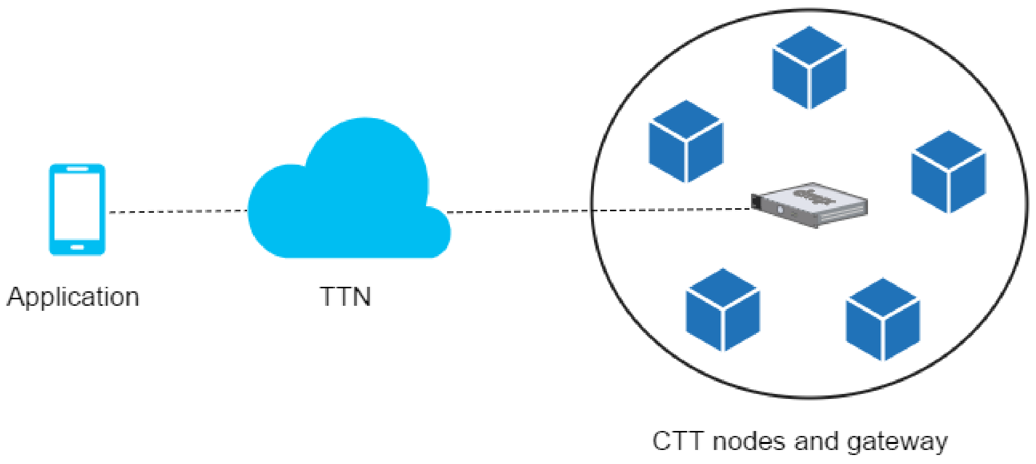


Figure 4.3: An application accessing sensor data through TTN (Made in `https://www.draw.io/`).

This architecture forms the core of what the project provides for participating cities. Figure 4.4 illustrates this architecture. It is a high level architecture intended to allow switching out components as necessary. Two prototype instantiations of this architecture have been employed in the Norwegian city of Trondheim and the Danish city of Vejle.

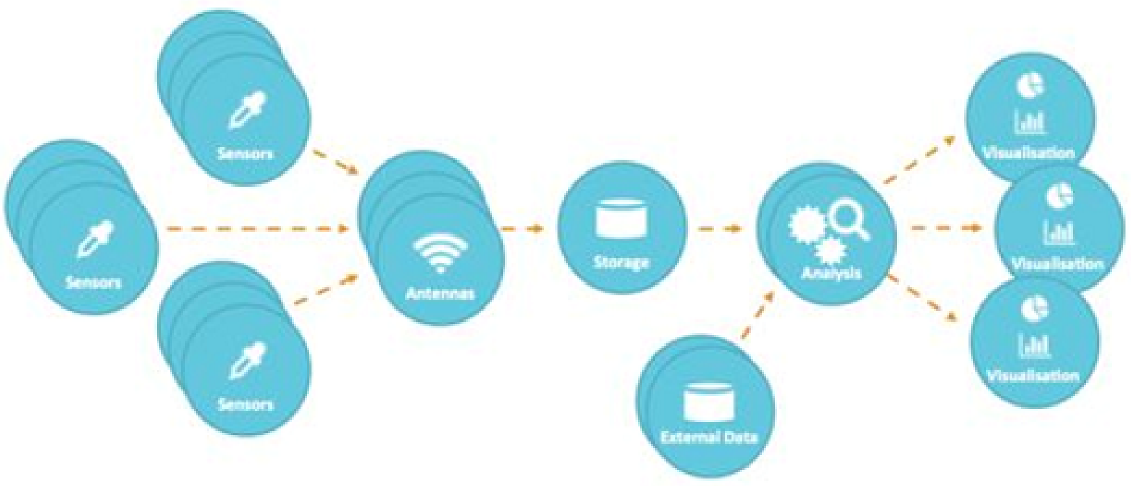


Figure 4.4: Architecture of CTT (source: CTT deliverable 2.4 [5]).

CTT also connects with other climate-focused projects. The carbon*n* Climate Registry (cCR) is a global project that allows local governments to measure and report their climate actions. The goals of cCR are to improve the standards of measured data and to encourage governments to report their climate actions to reduce GHG emissions [16]. The way that CTT sends their data to cCR is through the ClearPath tool. ClearPath is a reporting tool that provides a standardized way of reporting emission inventories [17]. It is also fully integrated with the cCR project, ensuring that reported data is compliant with cCR standards.

CTT builds upon previous efforts involving sensor technology. One study provided a large scale overview of the many modeling and assessment practices around Europe, concluding that there was a wide variety of types and quality regarding models of emission data, and that a standard way of doing this was needed [18]. Another relevant study reviewed some of the many modern sensing technologies available for air quality monitoring and provided comparisons of them [19]. CTT builds on modern studies such as these and was considered a good case of what should be able to be represented by the notation. It is a case that involves WSN, IoT, and geographically dispersed systems, which were all aspects that the modeling notation needed to support. CTT also has a defined architecture with specific physical objects that make up the basic architecture. The objects have resource limitations which is typical in IoT systems. The case also involves various data interfaces and stakeholders who can access the data, as well as frequent data movement. All these factors are part of what makes the CTT case representative of an IoT environment and what makes it a relevant case to model conceptually. By using CTT as a starting point for conceptual modeling, ideally it would result in a notation that is capable of representing all the relevant aspects of the case, as well as the potential to extend the notation for modeling other similar cases.

CTT has been ongoing simultaneously to this thesis. Models related to this project have been created and discussed with a researcher involved in the CTT project, as a means of evaluating the developed notation. The revised models can be found with explanations in Appendix A. The evaluation itself is described in Chapter 6.

CHAPTER 5

IMPLEMENTATION OF NOTATION

The proposed notation integrates aspects of BPMN and DFD. Process modeling and data modeling complement each other during the analysis phase when determining requirements, as highlighted in green in Figure 5.1 [20]. The modeling notation in this thesis provides an integrated notation that includes both process and data modeling. A combined focus on data flow modeling and process modeling makes it better suited for focusing on the data transformation processes and distributed structural elements in the system. The notation is meant to be used at the analytic and conceptual level for facilitating system analysis and requirements elicitation.

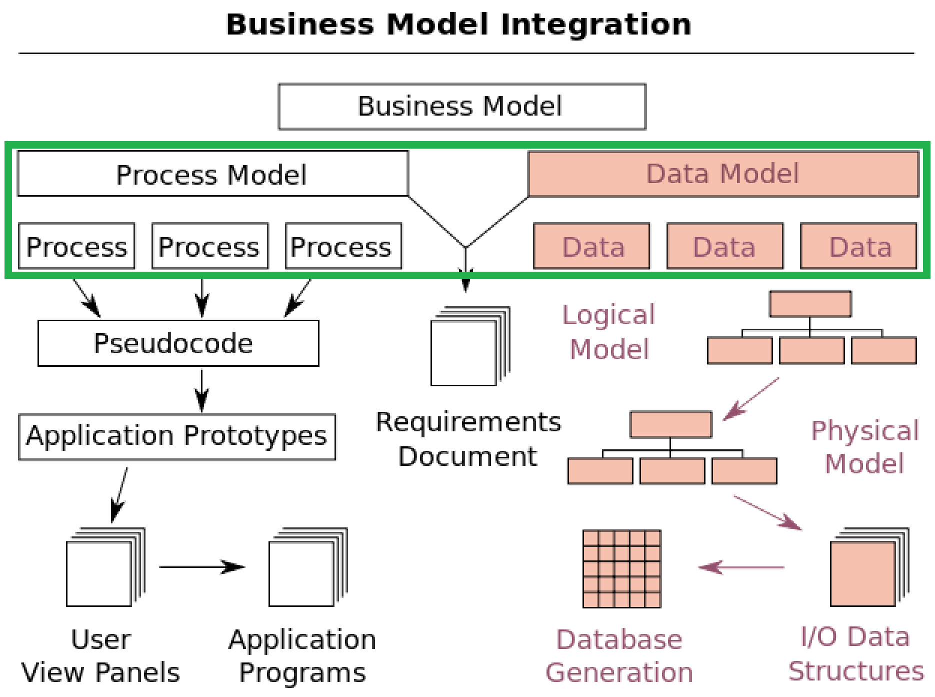


Figure 5.1: Integration of data modeling and process modeling at the conceptual level[21].

The notation also extends the existing notations with concepts that are relevant to the domain of mobile systems and IoT. Tolvanen & Kelly [22] discuss that in the case of integrating languages, often when integrating multiple languages in a single language, the resulting language becomes too unwieldy, thereby increasing the complexity. On the other hand, it allows the modeler to express many aspects in one view instead of having to create separate views due to no integration. There is a trade-off between complexity and expressiveness to consider when making the choice for a new notation. A single, fully integrated model, versus separate models that reference each other. The way that the notation proposed here deals with this issue is by using a restricted subset of elements from BPMN, integrated with a limited set of IoT-related constructs. This approach limits complexity while attempting to maximize expressiveness, for the purposes of the mobile domain.

The developed notation allows for explicit representation of data flow and data transformation among distributed constructs in a mobile system. The constructs that can be represented are specific to the mobile field, such as sensors, actuators and constructs that filter or aggregate data flow from various sources. The constructs borrowed from the BPMN notation generally retain their original use, while also being extended to interact with the domain specific constructs developed for this notation.

5.1 Creating a unified notation

By integrating the two notations of BPMN and DFD, a single notation is proposed here which covers aspects of both these notations. The notation is based on existing constructs from the two notations, as well as a number of new constructs which address the specifics of mobile information systems. This chapter explains origins and usages for all the constructs that are a part of the notation. For practical usage examples of the constructs, refer to Appendix A, where these constructs were used to create conceptual models of the CTT case.

5.1.1 New constructs

sensor and actuator

File reference:
(actuator) http://xml.master/xml/object_types/actuator.kmd
(sensor) http://xml.master/xml/object_types/sensor.kmd
(sensor actuator) http://xml.master/xml/object_types/sensor_actuator.kmd

Sensors and actuators are standard IoT entities. Their purpose is to observe the environment (sensor) or act upon the environment (actuator). In short, sensors are sources of data and actuators are destinations of data. This leads to a restriction on these entities; sensors may only have outgoing data movement and not incoming data. Actuators work in the opposite way, as they may only have incoming data movement and not outgoing ones. The representation for sensor and actuator is an entity marked by a symbol in the upper left corner. For sensors, this symbol is a circle with an outgoing arrow, while for actuators the symbol is a circle with an incoming arrow. In addition, a third entity type exist called Sensor Actuator. As the name suggests, this entity has the properties of both sensors and actuators, i.e. it can be used as both a source and destination for data. Figure 5.2 shows the sensor, actuator and sensor actuator entities. An example of what could constitute a sensor actuator is a thermostat, which can be responsible for both sensing and controlling the temperature of a system.

Sensors and actuators are included in the notation because these objects often exist in distributed mobile systems that involve data processing. Especially in the case of IoT where physical devices have to interact with software and the Internet, sensing and acting upon the environment becomes even more relevant. The symbols used to represent these constructs are intended to be simple. The circle with an outgoing arrow represents a source of data which is a sensor. Similarly the circle with an inbound arrow represents a destination of data which the actuator is. Following this logic, the sensor actuator construct is represented by a circle with both outgoing and inbound arrows. These symbols are similar to the ones used in BPMN4WSN[12]. BPMN4WSN does not make a distinction between sensors and actuators, while in this notation a distinction is made based on the function of the object.

Notes:

- A sensor is a source of data flow (1-n outgoing flows). However it may not be the destination of a data flow.
- An actuator is a destination of data flow (1-n incoming flows). However it may not be the source of a data flow.
- A sensor actuator is either a source of data flow (1-n outgoing flows), or the destination of data flow (1-n incoming flows), or both.
- A sensor may exist inside a sensing unit object, and in this case the sensing unit manages the sensor and is responsible for sending out the sensor's acquired sensor data.

Attribute name	Description / Usage
Name	A short text which describes what the sensor / actuator / sensor actuator is.
Cost	The monetary cost of this unit.
Expected lifetime	How long the unit expected to function properly, typically expressed in years or months.
Detected properties (sensor)	Lists the properties that the sensor can detect (gases, temperature, air humidity, etc).
Detection range (sensor)	Describes the range that the sensor can detect properties within. Typically expressed in parts per million (ppm).

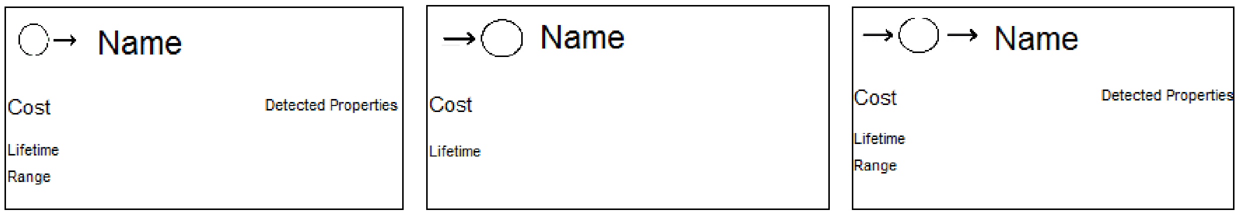


Figure 5.2: From left to right: Sensor, actuator, sensor actuator

Replicated entities

File reference:
(replicated actuator) http://xml.master/xml/object_types/replicated_actuator.kmd
(replicated sensor) http://xml.master/xml/object_types/replicated_sensor.kmd
(replicated sensor actuator) http://xml.master/xml/object_types/replicated_sensor_actuator.kmd

Replicated entities represent multiple instances of the same type of entity, e.g. multiple sensors distributed over a limited geographical area. This can be used for sensor and actuator objects. The representation of multiple instances of an object is an overlapping stack of boxes behind the construct. This way of representing replicated entities is similar to an existing project where replication was used in modeling complex choreographies in BPMN [11]. By utilizing replicated entities, it keeps the model cleaner and less complex because it eliminates the need to model the same concept multiple times. Figure 5.3 illustrates the replicated entities available in the notation.

This way of representing multiple instances is additionally sometimes used in network diagrams and data models, such as for a group of computers connected to a wireless router. The stacked boxes make it clear that there are multiples of something, and the box at the front gives the description of what they represent. In a mobile setting, there are many examples where there are multiple instances of something. It could be a group of sensors in an enclosed geographical area, a cluster of servers that all process incoming weather data, or many other things.

Notes:

- A replicated sensor has the same properties and connect rules as a regular sensor.
  - The same applies to replicated actuator and replicated sensor actuator.

Attribute name	**Description / Usage**
Name	A short text which describes what the replicated sensor / actuator / sensor actuator is.
Number of items	A numeral representing how many single objects make up the replicated object.
Cost	The monetary cost of each unit.
Expected lifetime	How long each unit expected to function properly, typically expressed in years or months.
Detected properties (replicated sensor)	Lists the properties that the sensors can detect (gases, temperature, air humidity, etc).
Detection range (replicated sensor)	Describes the range that the sensors can detect properties within. Typically expressed in parts per million (ppm).

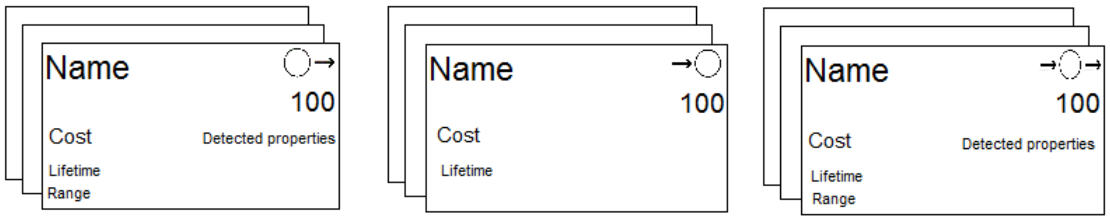


Figure 5.3: From left to right: Replicated sensor, replicated actuator, replicated sensor actuator

Sensing Unit

File reference:
http://xml.master/xml/object_types/sensing_unit.kmd

A sensing unit represents a node that contains a number of sensors or sensor actuators. Sensing units are responsible for transferring sensor data to other objects and managing the connected sensors. A sensing unit is identified by its hardware ID and/or its MAC address. It is also possible to specify the communication protocol used and the number of ports on the node. Figure 5.4 illustrates a sensing unit.

The visual symbol of the sensing unit is split into two halves. The upper half contains all the properties of the sensing unit while the lower half is the container part of the symbol, which is where sensor objects should be put. It is also possible to add BPMN processes to the bottom part of the sensing unit, which is relevant for cases such as showing how a sensing unit changes its output interval when it receives instructions over the network.

Notes:

- A sensing unit can contain sensor, sensor actuators, as well as the replicated varieties of these objects.
- The amount of objects inside the sensing unit should not exceed the number of ports specified on the properties of the sensing unit.

Attribute name	**Description / Usage**
Name	A short text that describes what the unit is.
Hardware ID	The unit's hardware ID. Can be used to identify the unit.
MAC address	The unit's MAC address. Can be used to identify the unit.
Protocol	The communication protocol used by this unit.
Number of ports	The number of ports restricts how many sensors can be connected to this unit.

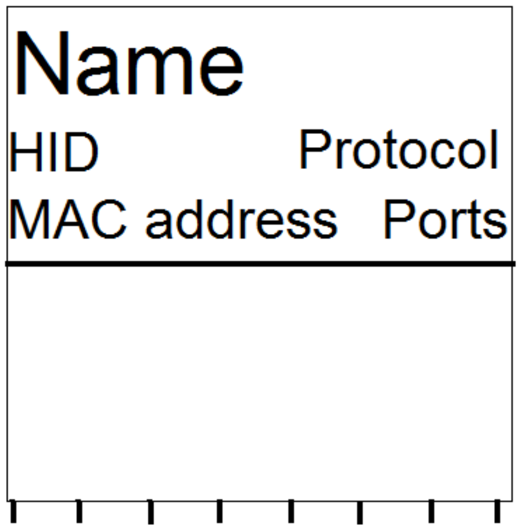


Figure 5.4: Sensing Unit

Data Flow

File reference:
(data flow) http://xml.master/xml/relationship_types/data_flow.kmd
(data flow timer event) http://xml.master/xml/relationship_types/data_flow_timer_event.kmd

Data flow represents the “flow”, or movement of data, as in standard DFD. It is used for data from physical entities like sensor and actuators, as well as data objects. This is the main relationship used to show how data moves throughout an interconnected mobile system.

A second type of this relationship is called *Data flow timer event*. This functions the same way as a standard data flow, with the addition of a timer event at its source and the possibility of defining an interval, which is how frequently the data flow gets captured along this relationship. The timer event symbol used on this relationship is the same symbol as a timer event in BPMN. Both kinds of relationships are shown in Figures 5.5 and 5.6 respectively.

Notes:

- The data flow relationship can be used on the following objects:
    - Sensors / Actuators / Sensor Actuators
    - Replicated entities
    - Sensing units
    - Data objects
    - Aggregates and filters
- To show that data is being moved and transformed simultaneously, this relationship needs to be used in conjunction with the transformation relationship.

Attribute name	**Description / Usage**
From text	Text describing the source end of the relation. Default: “From”.
To text	Text describing the destination end of the relation. Default: “To”.
Details (data flow)	A block of text which may be used to provide more in depth detail of the purpose of this relation.
Interval (data flow timer event)	A time interval which defines the frequency of the data flow getting captured (e.g. 30 minutes).

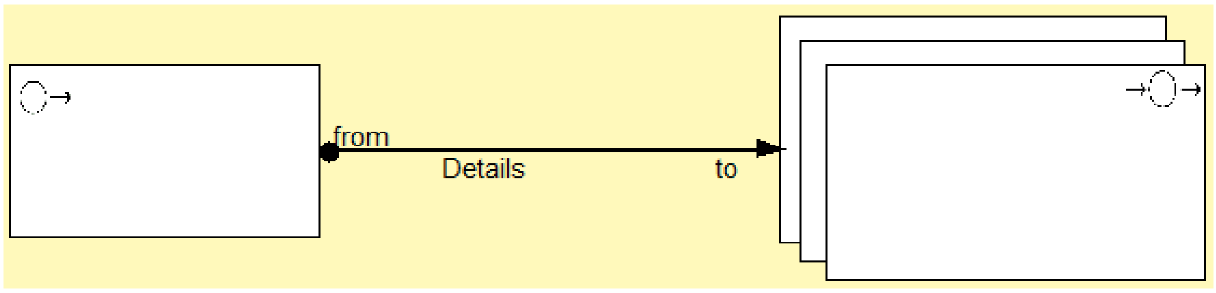


Figure 5.5: Data Flow relationship

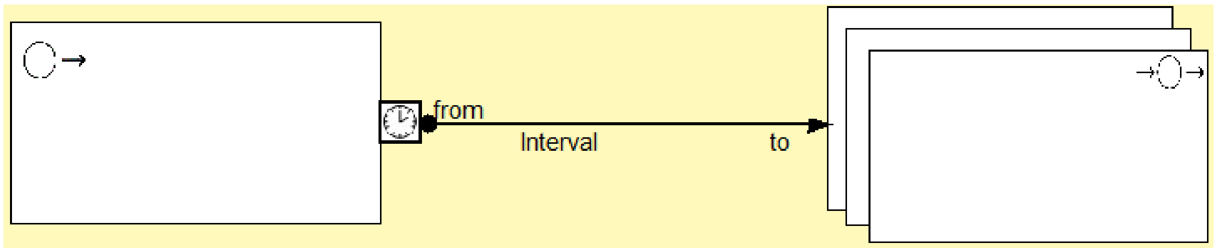


Figure 5.6: Data Flow relationship with timer event

Data object

File reference:
http://xml.master/xml/object_types/data_object.kmd

A data object represents a piece of data that can be transformed. It does not represent a physical object, but rather a bundle of related data, such as an hour's worth of emission data from a CO2 sensor. Typically a data object is related to one or more concrete entities which produce(s) that data. Allowed connections are data movement and data transformation from one data object to another, or from a data object to an entity. A data object may also exist within a BPMN swim lane as a replacement of the original data object that exists in BPMN 2.0. A data object may contain other data objects within itself to show that a big data set is composed of different kinds of smaller data sets. This type of structuring is frequently used in ArchiMate [23] for showing sub-parts of a whole. Figure 5.7 illustrates a data object.

A data object is represented as a rectangular box, similar to an entity, but made up of stippled lines instead of solid ones. The stippled lines used for this construct are sometimes used in other modeling languages for showing weaker relations and temporary objects. An example of this is in BPMN where stippled arrows are used for association relationships and stippled boxes are used to show groups of tasks. Similarly the data object construct is a bundle of data which normally is moved or transformed into something else in a mobile system.

Notes:

- A data object may be a source of transformation flow (0-n outgoing flows). It may also be the destination of transformation flow (0-n incoming flows).
- A data object may be a source of data flow (0-n outgoing flows). It may also be the destination of data flow (0-n incoming flows).
- Transformation flow is used to show that the data is being processed or altered in some way, while data flow is used to show actual movement of the data. If both transformation and movement is required, both types of relationships may be used together.
- Data objects can contain smaller data objects to illustrate more detailed decomposition of data.

Attribute name	**Description / Usage**
Name	A short text which describes what the data object is.
Details	A block of text which may be used to provide more in depth detail of the purpose of this object.

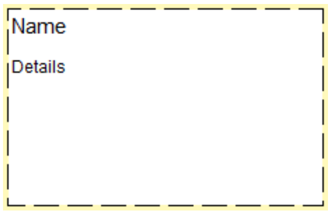


Figure 5.7: Data object

Data transformation

File reference:
http://xml.master/xml/relationship_types/transforms_to-transformed_by.kmd

A unique relationship arrow is used for showing data transformation. This is in addition to the data flow relation which has a different usage. While regular data flow shows movement of data, a transformation relationship means that the data itself is changed in some way. This arrow can be annotated with the cost of doing the transformation, e.g. by consumption of time, energy, or money. The relationship is represented by a stippled arrow with triangular dots on either end. The transformation changes one data object to another one. It may also be combined with a regular solid line to show that data is both moving and transforming. If it is desired to show the transformation process in detail, this can be done by having an outgoing transformation arrow to the swim lane containing the BPMN process and from the swim lane have an outgoing transformation arrow to the destination data object. Figure 5.8 illustrates a data transformation transforming one data object into another. Figure 5.9 shows a data transformation with a BPMN diagram in the middle, where the incoming arrow shows the original data and the outgoing arrow is the transformed data after the process is finished.

The motivation for this construct was that DFD originally is only able to represent data movement. With the addition of this construct, it is possible to separately visualize movement and processing of data. In mobile and interconnected systems, data tends to move and change rapidly and in complex ways. Therefore it is useful to have a way such as this to show the transformation processes separately from the movement of data. Graphically, the transformation arrow is designed to be clearly distinguishable from the regular data flow arrow at a glance, while still having the appearance of a typical relationship to avoid confusion.

Notes:

- This relationship can be used on the following objects:
  - Data objects.
  - Aggregates and filters.
  - swim lane diagrams and swim lanes.
- To show that data is being moved and transformed simultaneously, this relationship needs to be used in conjunction with the data flow relationship.

Attribute name	**Description / Usage**
From text	Text describing the source end of the relation. Default: “Transforms to”.
To text	Text describing the destination end of the relation. Default: “Transformed by”.
Cost / Description	A block of text which may be used to provide more in depth detail of the purpose of this relation, or the cost of performing the transformation.

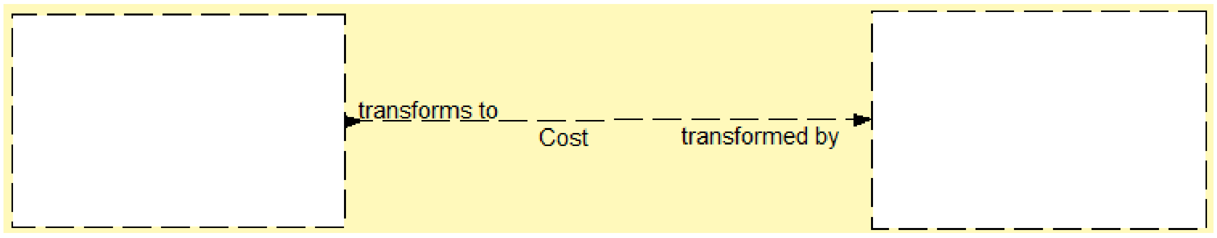


Figure 5.8: Data transformation relationship

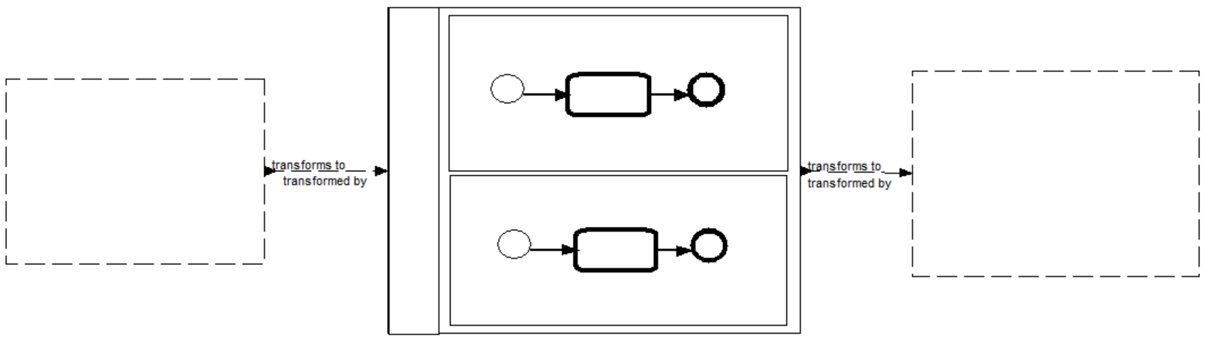


Figure 5.9: Data transformation with BPMN diagram

Location

File reference:
http://xml.master/xml/object_types/location.kmd

Geographical location can be represented on a high level by this object, a stippled box annotated with the location name in the upper left corner. It is a container object, and may be used to hold any other object in the notation. This construct encourages grouping of elements according to location. The purpose of this construct is to improve visual clarity and grouping of related entities. It does not impose any restriction on the entities within the box. It is only a high-level construct, and can not directly interact in any way with other constructs. Figure 5.10 shows a location container.

This way of modeling location as a pool was discussed as one of several alternatives for modeling of location in (Krogstie, 2012, p.256) [14]. Using containers to denote grouping can also sometimes be seen in UML use case modeling [24], In addition, BPMN has a "group" construct that is used to show which tasks are related without imposing any restrictions on the tasks within the group. That same idea is used for this location construct in order to be a construct only for visual clarity and no limitations on the objects within. The way that location is modeled in ArchiMate is very semantically similar to this as well, where a location is container for other objects and represents a conceptual point in space [23].

Notes:

- Location is a container that can fit any number and any type of other object.
- Locations may contain other locations, such as a research facility which has multiple departments within it.
- The location itself does not accept any incoming or outgoing relationships. The objects within the location are the ones that interact because the location is only a container.

Attribute name	**Description / Usage**
Name	A short text which describes what the location is.
Facilities	A block of text which list all the facilities available at this location.

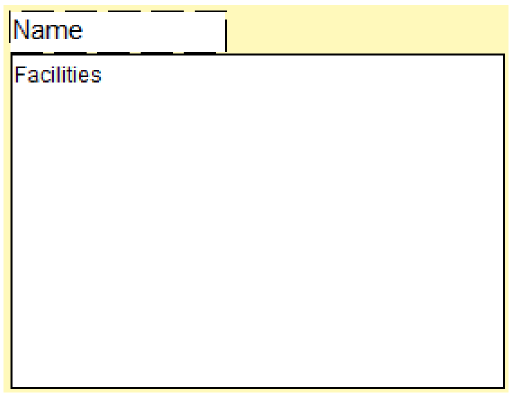


Figure 5.10: A location container

Aggregate and Filter

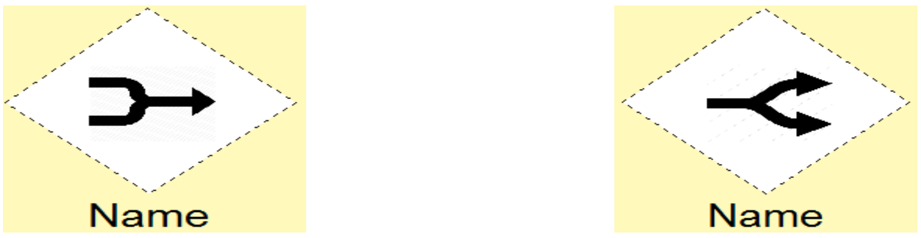
File reference:
(aggregate) http://xml.master/xml/object_types/aggregate.kmd
(filter) http://xml.master/xml/object_types/filter.kmd

Aggregate and filter objects are used to control the data flow. An aggregate is an object that accepts multiple input data flows (2-n) and aggregates them into a single output data flow. Conversely, a filter is an object that accepts a single input data flow and filters it into multiple output data flows (2-n). Both objects can also be used in this fashion for the transformation relationship.

The Aggregate symbol shows a stippled diamond containing an arrow. The arrow consists of two branches joining into a single point, to signify the way an aggregate object merges separate sets of input data into a single output. Conversely, the filter object shows a single arrow branching out into two end points to show how a filter splits a set of data into multiple output streams. The objects look similar to gateways in BPMN, which is because they share similar functionality (gateways control process flow, while aggregate and filter objects control data flow). These objects are illustrated in Figure 5.11

The aggregate and filter objects correspond to entities in the real world that control the flow of data, such as an antenna which aggregates signals from multiple sensors, or a data analysis tool which filters out relevant information to make visualizations of data. This means that these objects perform actual work on the data flows, as aggregating and filtering data usually entails more than simple joining and splitting of data.

Attribute name	Description / Usage
Name	A short text which describes what the object is.



(a) An aggregate object (b) A filter object

Figure 5.11: Aggregate and filter objects

5.1.2 Imported constructs from BPMN

Some of the most commonly used BPMN 2.0 constructs are imported to the new notation. By keeping a subset of the original notation, the new notation stays in line with the original one and allows for modeling the same constructs. The basic elements of BPMN that have been used are briefly described below, as conceived by the OMG group [25]:

Events

File reference:
(start event) http://xml.master/xml/object_types/process_start.kmd
(intermediate event) http://xml.master/xml/object_types/process_intermediate.kmd
(end event) http://xml.master/xml/object_types/process_end.kmd
(timer event) http://xml.master/xml/object_types/process_timer_event.kmd
(message event) http://xml.master/xml/object_types/process_message_event.kmd

An event is an occurrence which has a trigger and/or an impact. The supported types of events are: Start, intermediate, end, message and timer respectively as shown in figure 5.12. Events function the same way as standard BPMN.

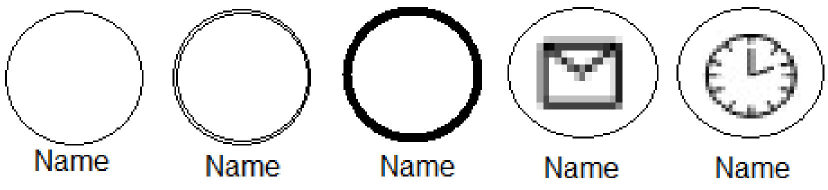
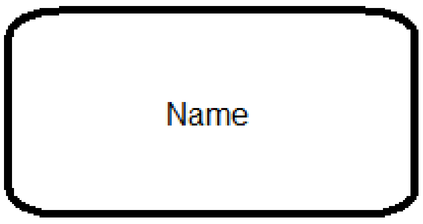


Figure 5.12: The five supported events

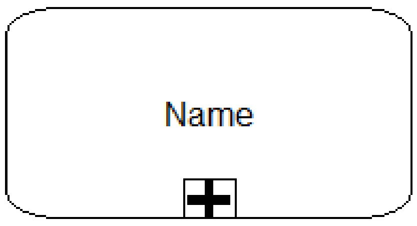
Activity

File reference:
(activity) http://xml.master/xml/object_types/process_task.kmd
(sub-process) http://xml.master/xml/object_types/process_subprocess.kmd

A process consists of activities, each represented as a rounded rectangular box. It represents a single activity of work to be performed. Activities can be tasks or sub-processes. These objects function the same as in standard BPMN. The sub-process object is a Metis container, which means it can be open or closed. An open sub-process reveals the details of the sub-process while a closed sub-process appears as in the above image. Figure 5.13 shows these objects.



(a) Process task



(b) Subprocess

Figure 5.13: Process task and subprocess

Gateways

File reference:
http://xml.master/xml/object_types/process_gateway.kmd

A diamond shape representing points of choice that control the sequence flows. Used for branching and merging multiple paths, and there exist many specialized types of gateways. Not all the specialized types were considered necessary for this notation, so only the basic types are implemented. As shown in Figure 5.14, the supported gateways are: exclusive gateway (follow a single flow based on condition), inclusive gateway (follow one or more flows) and parallel gateway (follow all flows simultaneously).

Gateways may also be used on either data flow relationships or transformation relationships with the same function as they have in BPMN sequence flows, in order to provide flow control on any part of the model. This is different from the aggregate and filter objects in that a gateway does not perform any work, it can only split or join flows whereas aggregate and filters can perform changes to their flows.

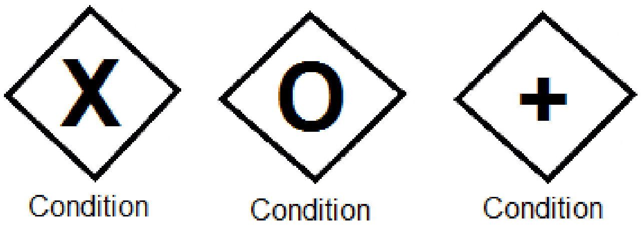


Figure 5.14: The three supported gateway types

Sequence Flow

File reference:
http://xml.master/xml/relationship_types/process_flow.kmd

A solid black line that represents the sequential order of activities in a process. This functions the same way as in standard BPMN, with the addition of a “Details” property for describing the relationship. Figure 5.15 illustrates this relationship.

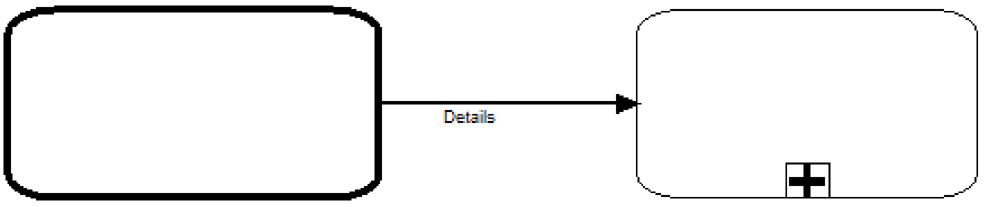


Figure 5.15: Sequence flow

Message Flow

File reference:
http://xml.master/xml/relationship_types/message_flow.kmd

A stippled line that represents message flow between tasks or swim lanes This functions the same way as in standard BPMN, with the addition of a “Description” property for describing the relationship. Figure 5.16 illustrates this relationship.

Additionally, a message flow may be used to connect data objects with swim lanes or individual BPMN tasks. This signifies that the source data objects are used in the destination processes or tasks. If used in reverse, it signifies that a source process or task modifies the destination data object in some way. The advantage of being able to show specific relationships on an individual task level like this is that it is possible to represent exactly which data object is used in specific parts of a process, and in which parts of the process data is processed.

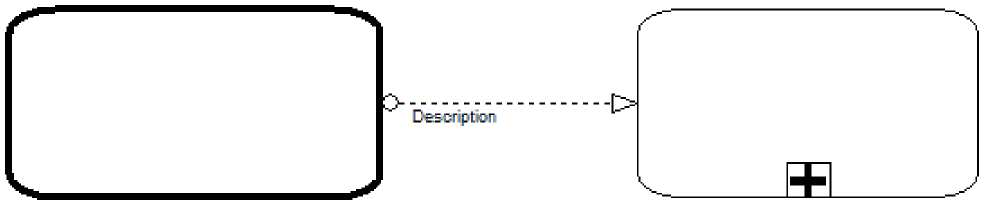


Figure 5.16: Message flow

Swim lane diagram and swim lane

File reference:
(swim lane) http://xml.master/xml/object_types/swimlane.kmd
(swim lane diagram) http://xml.master/xml/object_types/swimlane_diagram.kmd

A swim lane diagram contains one or more swim lanes, as in standard BPMN. Within the swim lanes, other BPMN objects may be used to describe the processes performed within. Additionally, non-BPMN objects such as sensors and data objects may not exist within swim lane diagrams. This is to stay true to BPMN as being a process modeling notation, therefore not including structural objects. Such objects may still interact with BPMN tasks and diagrams by using relationships.

A swim lane diagram exists as part of a larger model. The swim lane diagram in this notation may have incoming and outgoing data transformation relations, denoting that the swim lane diagram transforms the source data object(s) to the destination data object(s).

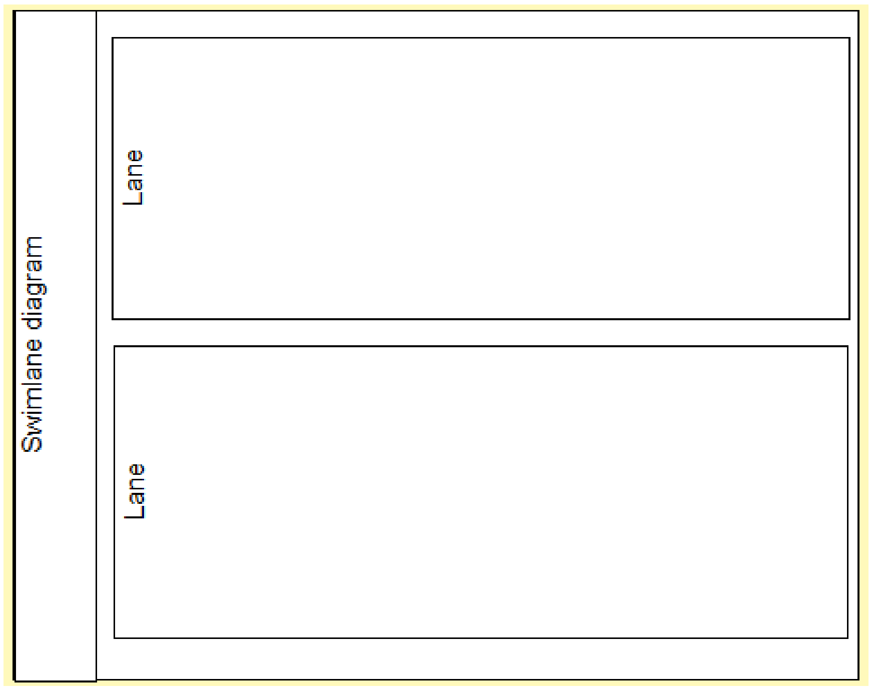


Figure 5.17: Swim lane and Swim lane diagram

5.2 Other constructs

Actors in a process are represented in the same way as in BPMN, where an actor is represented by a lane in the swim lane diagram. Examples of what constitutes an actor are a person or an organization. The swim lane construct is kept for processes while the naming convention is added to DFD entities.

Overlapping aspects are merged to remove redundancy, e.g. processes in DFD are replaced by BPMN processes which have a higher level of detail and make the DFD processes unnecessary. The process can be attached to entities or started by data transformations. The BPMN process entity is used much in the same way as a DFD process, i.e. it can be initiated by other entities/actors or data transformations. The BPMN model is used to give a detailed overview of the data transformation or data creation processes that exist in the mobile information system.

Data storage objects exist in both notation with the same conceptual meaning, and in this notation the more general *Data object* construct is used instead. This object can represent data storage, temporary data, and decomposition of data all with the same object, rather than having specified object types for specific types of data. This could be considered both an advantage and a disadvantage. While the use of a single object reduces complexity and allows high level modeling, it removes the possibility of differentiating between types of data.

CHAPTER 6 EVALUATION

This chapter evaluates the notation that has been developed, focusing on the evaluations that were done regarding the CTT case through discussions with a lead researcher on this project. Following this is an evaluation of the notation using existing model evaluation frameworks. A number of frameworks exist for evaluating the quality of modeling languages and conceptual models. As there does not exist a single standard framework used in practice, this evaluation has been approached by using two separate frameworks with different quality metrics, in order to conduct a well-rounded evaluation.

6.1 Empirical evaluation

After an initial version of the notation was created, some tests were conducted, attempting to model parts of the CTT case. These tests illuminated some things to be added or changed in the notation. This section discusses some of the findings of this testing as well as what measures were done with the results.

After an initial model of the case was created, the model was discussed with Dr. Dirk Ahlers from the CTT project. During this meeting, some limitations of the notation were discovered. Sensors were at that point modeled as individual structures, existing on their own. However this was inaccurate to reality as sensors exist as part of sensing units that manage all the sensors connected to them. Because the sensing units were an integral part to the systems and because there was no way to adequately show them with the existing constructs, a new specific construct *sensing unit* was created for this.

Some relevant properties of the objects were also discussed, such as sensing units being identified by their hardware ID's or MAC addresses, as well as the different properties of the individual sensors. The notation was refined to account for the relevant properties. The notation was also made more flexible to allow certain connections that were previously impossible. An example of this was allowing data objects to connect directly to individual BPMN tasks which use that specific data object, rather than just having a general transformation relationship from the data object to the BPMN diagram. Another example was allowing BPMN diagrams to exist inside sensing units, thereby showing details of the relevant processes in these units and being able to show how the sensing units address the feedback data they receive from other parts of the system. The original BPMN diagrams used in the models were too basic to provide an analysis on them, which was a problem at the model level rather than at the metamodel level. Following the meeting, these parts of the model were expanded and the BPMN diagrams were coupled more tightly with related external objects to make the overall model more cohesive.

6.2 SEQUAL

In (Krogstie, 2012) [14], a framework for how to evaluate the quality of conceptual modeling languages for performing a modeling task is described. This framework is called SEQUAL and evaluates the modeling language based on the perspectives of the domain, comprehensibility, participants, modelers, tool, and organization.

Domain appropriateness

Domain appropriateness evaluates the modeling language's ability to express all the objects in a certain domain. In this case, the domain was IoT, also including sensor networks. The ideal was that the language should be powerful enough to express everything within the domain. However in this case, the IoT domain has a massive amount of distinguishable objects. Therefore the evaluation focuses primarily on the parts of the domain that have been considered useful for the modeling task at hand, which was the CTT case in this project. To achieve this, parts of the modeling notation were designed after the CTT architecture shown in Figure 4.4, as well as meeting with a researcher on this project to reveal other constructs important to represent in the notation.

Comprehensibility appropriateness

Comprehensibility requires that the actors involved in the modeling effort are able to understand the language. To accomplish this, the amount of objects in the notation should be kept at a reasonable level, while keeping the language flexible in regards to amount of detail shown in models. The objects must also be easily distinguishable from each other. In section 6.3, the notation is evaluated more in detail from empirical and cognitive principles that are highly related to the concept of comprehensibility. At a high level, the effort was focused on keeping a limited set of objects that were distinguishable while still being explicit enough to fulfill the modeling task at hand.

Participant appropriateness

Appropriating the language towards the participants' knowledge means that the conceptual basis of the language must correspond as much as possible to the participants' perception of reality. In the case of a domain specific language, the constructs are very specific in meaning and usage, as they are related to things within the same domain in the real world. If the participants who are to use these models are already familiar with the target domain, it may make the notation more intuitive and easier to learn.

The notation used parts of the BPMN notation. BPMN is widely used for process modeling in enterprises and is therefore regarded to be well understood. Participants who have previous knowledge of BPMN, should find the models easier to understand in such cases as it cuts down the amount of objects that need to be learned significantly.

Modeler appropriateness

Most of the points discussed in Participant appropriateness also apply to modeler appropriateness. This principle also requires that the modeler is able to express all their knowledge in the notation. To facilitate this, objects in the notation were variable in how much detail could be expressed with them.

Tool appropriateness

Tool appropriateness requires that the notation lends itself to automatic reasoning or execution. This notation's main purpose was to be used for conceptual and analytical purposes, and was therefore primarily aimed at manual techniques involving stakeholders, rather than supporting executable environments. If automatic reasoning is to be a future prospect of the notation, a higher degree of formalization would be required first.

The tool used to develop the notation was Metis by Troux architect[15]. This tool contains most of the basic functions necessary to quickly implement constructs and add properties to them. This was useful for the purposes of making a prototype notation as new constructs were needed throughout the project timeline and the existing constructs frequently needed to be changed. On the other hand the tool was difficult to use for some advanced purposes such as changing an object's symbol based on its property values. This, accompanied by the fact that the built-in symbol editor was very basic compared to modern graphic design software, made the tool seem somewhat outdated overall. Section 6.4.2 describes these Metis-related constraints in more detail while Chapter 7 discusses the impact that the tool limitations had on the notation itself.

Organizational appropriateness

The modeling language should conform to the modeling methods and standards of the organizations that are going to use the notation, which leads to the principle of organizational appropriateness. One way the notation lent itself to organizational standards was by using BPMN constructs, which is already a leading standard for process modeling. Additionally, by using the CTT case as a target case, this made it possible to hold discussions on how to make the notation fulfill the organizational needs of the target case. A weakness of the notation was that it had been implemented in the Metis tool, which is not widely used. However this weakness could be considered negligible in this case because the notation was primarily an experimental prototype, as mentioned in section 3.1. Because of this, the Metis tool was considered acceptable for the purposes of this project.

6.3 Moody

Moody [26] has proposed nine principles for visual notations. These principles were derived from empirical evaluations as well as other disciplines such as cognitive psychology, linguistics, semiotics and graphic design. Following the principles makes it possible to test and evaluate visual notations, and others have already applied this for testing the cognitive effectiveness of BPMN [27]. The notation proposed in this thesis has also been evaluated using these principles, which are briefly described in Table 6.1.

Name	Meaning
Semiotic Clarity	Having a 1:1 mapping between concepts and symbols of the notation.
Perceptual Discriminability	Different symbols should be easily distinguishable from each other.
Semantic Transparency	Symbols should be semantically related to its meaning so that it is intuitive to understand.
Complexity Management	The notation should be able to have ways of reducing complexity in diagrams, such as abstracting parts of the model into views.
Cognitive integration	The notation should make it easy to navigate between diagrams.
Visual Expressiveness	Using a variety of visual constructs in the notation.
Dual Coding	Using both text and visual constructs in an integrated manner so they complement each other.
Graphic economy	Reducing the amount of different symbols to keep the notation simple. A high number of different symbols can make the model cognitively difficult to interpret.
Cognitive fit	Making the notation adapted to different tasks, and for different audiences.

Table 6.1: Moody's nine principles

The following is an overview of how the developed mobile notation fulfills these nine principles:

1. Semiotic Clarity: Every symbol in the notation is distinct in meaning and appearance. No concept has multiple symbols, and conversely no symbol has multiple meanings. Due to the notation being domain specific to the field of mobile and IoT systems, the modeling constructs represent very specific objects from real life, such as a sensor. This makes it less likely that different constructs overlap due to their correlation with real objects. Despite this, there may be real constructs that cannot be represented in the notation due to this specificity. In that case there would be a *symbol deficit* in the notation which would require the addition of new elements that cannot currently be represented. Some examples of potential symbol deficit in the current notation are non-sensor networked devices that are a part of the IoT, such as vehicles and smartphones. These are just some examples of things with embedded devices that make up the IoT and might be relevant to include in models. Another possible addition would be explicit symbols for the actors involved, which could include organizations as well as individual persons. Actors can not be explicitly modeled in the current notation outside of processes even though visualizing them may be useful in some situations.

2. Perceptual Discriminability: Moody argues that visual differentiation is more efficient than textual differentiation between objects. This is realized in the notation as all the main types of objects can be differentiated by their shapes, line types, or graphical symbols. However some similar objects such as sensors and actuators may require some experience to tell apart at a glance. These objects look the same but can be told apart from the icon denoting their use. A similar situation exists with aggregate and filter objects.

3. Semantic Transparency: Symbols have been frequently used to guide the user towards understanding the constructs. Examples of this include sensors which show a symbol of incoming data, actuators showing outgoing data, aggregating objects showing converging flows, and filters showing separating flows. Physical objects typically are modeled in solid lines, while anything involving abstract objects like data objects are modeled with stippled lines. The notation uses spatial overlap where possible to show subtypes or encompassment, which is something Moody argues can be a more semantically transparent mechanic than using object relationships.

4. Complexity Management: Location objects are used for logical groupings of objects, increasing the readability of the model. Some parts of the model can be modeled on a general level such as BPMN diagrams or subprocesses. The degree of details on the model can be varied, allowing the user to make more or less complex diagrams. However, the model can quickly get complicated when many objects are needed in the same view. An attempt to counteract this is to make use of the replicated entities of the notation rather

than showing each individual one. Also, making use of the spatial overlap reduces the number of relationships needed in a model, which may further reduce the complexity and clutter.

5. Cognitive integration: The Metis tool has some mechanics for breaking up models into separate views. This requires the modeler to first create the views and filling one with objects, then replicating the necessary objects into the second view. However one view is the source view and the other is the dependent view, meaning that the dependent view responds to changes in the source, but not the other way around. There does not seem to be a way of creating fully integrated diagrams in Metis, and creating multiple views from an existing model is laborious. It would be advantageous to reimplement the notation on a different platform that allows view creation in a more modern way. The notation itself also does not have any explicit ways of separating diagrams.

6. Visual Expressiveness: The notation attempts to use the most variety possible within the Metis 5.2 tool. Line types, shapes and symbols are used to distinguish the different constructs. Color is not used, which could have led to a higher degree of variety but could also cause a problem for color blind users, as discussed by (Krogstie, 2016, pp.111-117)[28]. Another point of note is that graphical encoding is preferred over textual encoding, which is realized in the notation as discussed in the point regarding Perceptual Discriminability.

7. Dual Coding: Text is primarily used to show properties or additional details on objects. Most of the text is optional, meaning the viewer can include as much or as little text on the models as desired. Some objects can be freely annotated with a descriptive text, and relationships have default text on them to supplement the meaning of the visual symbols.

8. Graphic economy: As this notation is domain specific, the objects tend to also be specific rather than general. Some generalized objects exist in the notation such as data objects which can be a variety of things. However if all objects of the IoT domain were to have a representative modeling symbol, there would be a case of symbol overload in the notation. Therefore the number of constructs in the notation is intentionally kept at a manageable number. This principle of graphic economy could become more of an issue if the notation were to be extended to be able to represent more concepts.

9. Cognitive fit: This principle suggests the need to develop separate dialects of the notation for novice and expert users. As this notation is specific rather than for general use, it is geared more towards expert users. The notation is certainly harder to learn for someone without BPMN experience, as a subset of BPMN is used in the notation. For users experienced with BPMN, the learning curve is gentler.

6.4 Constraints and deficiencies

All modeling languages have certain deficiencies, due to the fact that every modeling language has a focus and therefore is typically unable to model things outside its focus. Certain trade-offs must be made between maximizing the expressiveness of the language versus the simplicity and clarity in order to be understandable. This section will discuss the main deficiencies of BPMN, some of which have been covered by the integrated notation as well as potential new deficiencies that may have arisen. Regarding DFD, its main deficiency is that it lacks formal expressive power due to the low amount of constructs. The new notation covers this by extending it with new constructs for more specialized usage and adds restrictions to them, as well as introducing BPMN constructs to the notation.

6.4.1 BPMN constraints

(Krogstie, 2016, pp.209-218)[28] discusses some of the deficiencies of BPMN. BPMN lacks the power to represent states of objects. Without states, there is a risk of some business rules not being captured properly in the diagrams. BPMN also cannot properly represent the structure of things in a system, as it only covers business processes. The new notation needed to address this due to IoT being a focus point and system structure being important when data flow modeling is involved. Resources can also not be modeled properly in BPMN. This is another relevant point for the new notation, as resources are present in IoT and IoT objects often have limitations on available resources such as cost and sensor range. These aspects should therefore be a part of an IoT modeling notation.

BPMN contains many advanced concepts and has multiple representations for similar things. The sheer amount of constructs causes significant complexity if trying to understand every concept in the notation. This issue is mitigated in the new notation because only a basic subset of BPMN is used. This raises the possibility that the subset used in the new notation lacks the expressiveness that standard BPMN provides. However [29] discusses how only a small portion of BPMN elements are used in practice, and how different subsets are common to find. The study includes a frequency chart showing how commonly each BPMN object was found to be used in practice. Only 19 objects showed an occurrence of at least 25% in the study, and the subset used in this thesis covers 16 of those. The three objects not covered by the subset are association relationship, text annotation, and end terminate event. An analysis done by Moody on the BPMN notation[30] built further upon this and concluded that the graphical complexity of BPMN is a major issue for its general usability. Moody also argued in favor of using subsets of the notation for analysis tasks. These studies indicate that using a representative subset of BPMN actually leads to more benefits than disadvantages, as has been done in this thesis.

6.4.2 Mobile notation constraints

The developed notation is fairly specific in what it is able to represent because of the focus on IoT. Due to this specificity, it may limit the possibilities of extending the notation for other purposes than the one originally intended. The modeling effort in this thesis has been predominantly focused on the CTT case, and thus the notation would need further testing to discuss how its usefulness could extend to other projects. Effort has been put into making the notation general rather than specific wherever possible, however some trade-offs needed to be made in order to represent the necessary parts of the CTT case. One instance of this is the sensing unit object, which was discovered to be important during an evaluation meeting. Thus since there was no way to represent something like a sensing unit at the time, it was implemented in the notation for the express purpose of being able to include it in the case model. This was useful for modeling the CTT case, however it also increased the graphic complexity of the notation and may not be useful for modeling other cases.

Some objects could benefit from more specialization. One example of this is the data object, which is a general representation of abstract data. It is currently not possible to differentiate between different types of data, which could be relevant when dealing with sensitive data that has multiple levels of security clearance, or for differentiating between data in transit and data at rest.

Some constraints are a result of limitations with the tool used. As mentioned in the tool appropriateness section, Metis was useful for making a prototype notation, but had some issues related to it being outdated. Some of the main issues found are listed below, while the impacts of these issues are discussed in Chapter 7:

- The tool was difficult to use for some advanced purposes such as changing an object's symbol based on its property values. The tool did support implementing methods that could actively manipulate the objects, but the way of implementing this was convoluted and not very user friendly.
- The built-in symbol editor was very basic compared to modern graphic design software. It lacked ways to adjust specific shapes and sizes in a nice and consistent way. Most of the symbol creation had to be done in a manual way. Additionally when creating relationships there were a limited number of line types and ways to represent the arrows.
- While creating the models themselves, resizing objects often had adverse effects on the text and symbols on the objects. This sometimes led to mangled graphics and text that was too small or too big. This was especially bad on objects that contained custom symbols, as the symbols would often look disfigured when being scaled up or down.

- Metis contains existing notations such as BPMN which can be used in modeling and metamodeling. However, mixing and matching objects from existing notations and trying to create new objects that interact nicely with these existing objects, proved to be difficult to do. Most existing objects were tightly connected to other objects from the same notation by their design and usage, which made it impractical to export individual objects from BPMN to the new notation.

CHAPTER 7 DISCUSSION

This chapter discusses how the developed modeling notation solves the problem stated in the introduction, while interpreting the results and putting them in context with the fields of IoT and modeling in general.

In every modeling notation, a compromise has to be made between adding enough constructs to make the notation able to express as many cases as possible, but keeping the number of constructs concise enough that the notation is understandable and easy to use. For the mobile notation, the focus has been on being able to model sensor networks and the choices of constructs were been made based on the CTT case previously discussed. For the reasons discussed in Chapter 3, a single case was focused on in the initial version, and the notation can be applied to additional cases hereafter. While the notation provides a base for modeling cases related to IoT and mobile systems, it can easily be further extended as needed to include more constructs. The notation could also be refined further to enable expressing more precise properties on the IoT objects, thus tailoring it more specifically to the case being modeled. The risk involved in doing this is that by tailoring the properties towards specific cases, the notation might lose its suitability for general purpose modeling of IoT systems.

The decision to include elements of BPMN in the notation was partly to keep the notation following standard practices. Being able to model business processes was central in the notation to show how the data transformation processes occur in detail, as well as other event-based parts of mobile systems. The BPMN constructs themselves were constructed from scratch in Metis, while adhering to the existing BPMN 2.0 notation. Some benefits of using the BPMN standards instead of creating entirely new constructs include:

- BPMN is a widely used notation in practice and by using it, experienced modelers will have fewer new constructs to learn. Therefore the notation should also be easier to learn and cause fewer modeling mistakes, assuming that the modeler has previous experience with BPMN.

- Time is saved on designing and conceptualizing the modeling constructs as they already exist in BPMN and simply needed to be recreated in Metis.

- The new modeling constructs have been designed to fit together with the BPMN symbols, both visually and functionally. In this way, BPMN functions as a base notation and all new constructs were designed on top of this standardized base notation. Ideally this would mean that the new constructs adhere to the style and standards of BPMN.

However, as discussed in Section 6.2 on the topic of tool appropriateness and in Section 6.4.2, there were limitations of Metis that impacted the final notation in some ways. One such way was that originally, sensors and actuators were intended to be a single object with a changeable symbol to indicate whether it was a sensor or an actuator. Modeling these as a single object would be logical due to their similar function and properties, but due to challenges in Metis regarding actively changing the symbols on objects, they were instead created as separate individual objects. This led to an increase in the amount of objects and therefore an increase in the complexity of the notation. Because of the simplicity of the symbol editor, some parts of the notation ended up being more similar to each other than intended. This was primarily a concern for the relationships, as there were four distinct relationship types in the notation (data flow, sequence flow, transformation flow, and message flow). The intent was for these to be clearly distinguishable from each, but because of limitations in the editor they may be more difficult to distinguish from each other than preferred. It is also worth noting that the graphical issues regarding the model creation make it somewhat convoluted to make visually appealing models. This is mainly a problem with how Metis scales objects, and it is certainly possible to make visually good models, although it often requires some fiddling where it could benefit from better automatic scaling. As discussed, the BPMN objects were also made from scratch, which was not the original plan. Originally the idea was to implement existing BPMN objects to integrate these with the new constructs. This was another difficult task to accomplish in Metis, so it was found easier to make the needed BPMN objects from scratch instead. This could be considered a benefit in the end, as it made the BPMN elements have the same visual style as other objects, and their function could be precisely tailored to the purposes of the notation. This was beneficial because in some cases the BPMN objects had their function slightly altered from their original use, or had a reduced functionality. As an end note regarding the Metis tool, it was quite simple to implement the basic constructs, symbols, and rules within it. This was the main justification for using it to begin with, as there were expected to be frequent changes and additions to the notation.

Previous modeling efforts that include IoT and sensor networks typically focus on smaller optimizations or extensions to existing languages. Some service-oriented approaches have also been performed, as discussed in Chapter 2. This notation was an attempt at conceptually modeling such systems on a larger scale, capable of expressing many different parts of IoT and mobile systems. This kind of modeling has become more relevant in the modern society where constantly more devices become interlinked in complex networks, over geographically dispersed areas.

The notation is still at an early prototype stage, and it has the potential to be extended and formalized by future research to allow a higher degree of expressive power. As it currently stands, its main contributions to the domains of IoT and mobile information systems are the specific visual constructs that make it possible to create structural and process oriented models, which facilitates the analysis of the systems being modeled. The next logical steps for the notation are more thorough empirical testing, as well as reimplementing and formalizing the notation on a more modern platform.

CHAPTER 8 CONCLUSION AND FURTHER WORK

This chapter concludes the project with a summary of the work up until this point and discusses potential future work.

This thesis has conducted an approach at creating a conceptual modeling notation that is able to represent key parts of IoT systems and sensor networks. the main motivation for this was that domain specific notations typically are able to represent objects on a higher level of details than general notations, and often more efficiently as well. The notation has been based on existing notations and standards used in practice, taking inspiration mainly from the process modeling found in BPMN and the structural modeling used in DFD. The project has used the CTT project as a case study for a system that was desirable to model with the notation. A set of specific and some general constructs were used to represent parts of this system with variable levels of detail. The notation has also been evaluated by using two general frameworks for modeling language quality, as well as empirical testing by discussing the models with a researcher on the CTT case. The notation could still be refined to enable expressing more precise properties on the IoT objects, thus tailoring it more specifically to the case being modeled.

A modeling notation can always be refined and extended to fit other cases and purposes than it has been previously used for. The same applies in this case, as the notation has been tested on several cases. These cases have mainly been related to IoT and sensor networks as this has been the focus of the notation, however it is conceivable that the notation can be applied to a number of other systems as well. Sensor networks exist in many areas of today's society, such as traffic systems, the Internet, weather radar networks, and so on. The notation could be modified and extended to model these types of systems, however the amount of objects have been kept at a manageable level in the initial version to make the notation easier to learn and reduce model complexity. Additionally, the model would benefit from being reimplemented in a more stable and widely available tool before any further standardization or refinement takes place. Metis was a useful tool for creating a prototype, but is too outdated for creating a more formalized modeling language used in practice. Model-driven system generation is also a possibility for future extension, such as in ThingML[10]. This was not done in the initial model due to the focus being on conceptual visual models for analysis purposes. For generating systems, a more formal structure would need to be imposed on the notation, like the approach in [13].

BIBLIOGRAPHY

[1] J. Visser A. van Deursen, P. Klint. Domain-specific languages: An annotated bibliography. *ACM SIGPLAN NOTICES*, 35:26–36, 2000.

[2] F. A. Kraemer F. Anthonisen J. Krogstie D. Ahlers, P. A. Driscoll. A measurement-driven approach to understand urban greenhouse gas emissions in nordic cities. In *NIK: Norsk Informatikkonferanse 2016*. NIK, 2016.

[3] J. Krogstie A. Wyckmans D. Ahlers, P. A. Driscoll. Carbon track and trace (ctt). `https://www.ntnu.edu/ad/ctt`, 2016-2017. Accessed: 10-05-2017.

[4] S.Y. Lim B. Weber C. Haisjackl, P. Soffer. How do humans inspect bpmn models: an exploratory study. In Editorial Board of SoSyM, editor, *Software & Systems Modeling*. Springer, 2016.

[5] P.C.V. Ramesh S.-J. Park V. Khatri, I. Vessey. Understanding conceptual schemas: Exploring the role of application and is domain knowledge. In *Information Systems Research 17(1)*, pages 81–99, 2006.

[6] W.-S. Rhee H.-S. Choi. Iot-based user-driven service modeling environment for a smart space management system. In L. Lavagno, editor, *Sensors*, Basel, Switzerland, 2014. Multidisciplinary Digital Publishing Institute (MDPI).

[7] J.-W. Lee J. Kim. Openiot: An open service framework for the internet of things. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 89–93, 2014.

[8] R. Vitenberg R. Rouvoy F. Eliassen K. Dar, A. Taherkordi. Adaptable service composition for very-large-scale internet of things systems. In *Proceedings of the Workshop on Posters and Demos Track (PDT '11)*, pages 11:1–11:2, New York, NY, USA, 2011. ACM.

[9] S. Jain W. Brunette R. C. Shah, S. Roy. Data mules: modeling and analysis of a three-tier architecture for sparse sensor networks. In *IEEE SNPA WORKSHOP*, pages 30–41, 2003.

[10] B. Morin F. Fleurey. Thingml documentation. http://thingml.org, 2011. Accessed: 15-03-2017.

[11] G. Decker & F. Puhlmann. Extending bpmn for complex choreographies. In Tari Z. Meersman R., editor, *On the Move to Meaningful Internet Systems*, Berlin, Heidelberg, 2007. CoopIS, DOA, ODBASE, GADA, and IS. OTM 2007, Springer.

[12] N. Oertel O. Kopp C. T. Sungur, P. Spiess. Extending bpmn for wireless sensor networks. In *2013 IEEE 15th Conference on Business Informatics (CBI)*, Vienna, Austria, 2013. IEEE.

[13] C. Magerkurth S. Meyer, A. Ruppen. Internet of things-aware process modeling: Integrating iot devices as business process resources. In Ó.C. Pastor C. Salinesi, M. Norrie, editor, *Advanced Information Systems Engineering: 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013.*, pages 84–98, Berlin, Heidelberg, 2013. Springer.

[14] J. Krogstie. Model-based development and evolution of information systems: A quality approach. pages 249–278, 327–388, London, 2012. Springer.

[15] Troux technologies. http://www.planview.com/products/troux/. Accessed: 15-03-2017.

[16] carbon*n* climate registry. http://carbonn.org/. Accessed: 28-05-2017.

[17] Clearpath. http://www.clearpath.global/. Accessed: 28-05-2017.

[18] P. Thunis et al. Overview of current regional and local scale air quality modelling practices: Assessment and planning tools in the eu. In M. Beniston, editor, *Environmental Science & Policy*, volume 65, pages 13–21. Elsevier, 2016.

[19] X. Querol N. Castell M. Viana M. C. Minguillón, C. Guerreiro. Real-world application of new sensor technologies for air quality monitoring. European Topic Centre on Air Pollution and Climate Change Mitigation (ETC/ACM), 2013.

[20] R. Sarfaty P.R. Smith. Creating a strategic plan for configuration management using computer aided software engineering (case) tools. In *Conference: 1993 National Department of Energy (DOE)/contractors and facilities data acquisition and control user‘s group meeting*, Livermore, CA (United States), 1993.

[21] via Wikimedia Commons By Process_and_data_modeling.jpg: P. R. Smith. Redrawn by M. D. Dekker Data_modeling_context.jpg: M. D. Dekker derivative work: Razorbliss [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)].

[22] S. Kelly J-P. Tolvanen. Integrating models with domain-specific modeling languages. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, DSM ’10, pages 10:1–10:6, New York, NY, USA, 2010. ACM.

[23] The Open Group. Archimate® 2.1 specification. `http://pubs.opengroup.org/architecture/archimate2-doc/`, 2012-2013. Accessed: 18-01-2017.

[24] Microsoft. Uml use case diagram guidelines. `https://msdn.microsoft.com/en-us/library/dd409432.aspx#UsingSubsystemBoundaries`, 2017. Accessed: 20-02-2017.

[25] Object management group Inc. (OMG). Business process model and notation (bpmn) version 2.0. `http://www.omg.org/spec/BPMN/2.0/`, 2011. Accessed: 07-09-2016.

[26] D. L. Moody. The"physics"of notations: Towards a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering 35*, 2009.

[27] D. Amyot N. Genon, P. Heymans. Analysing the cognitive effectiveness of the bpmn 2.0 visual notation. In *Proceedings of the Third International Conference on Software Language Engineering*, SLE'10, pages 377–396, Berlin, Heidelberg, 2011. Springer-Verlag.

[28] J. Krogstie. Quality in business process modelling. pages 111–117, 209–218, London, 2016. Springer.

[29] J. Recker M.Z. Muehlen. How much language is enough? theoretical and practical use of the business process modeling notation. In *CAiSE 2008: Advanced Information Systems Engineering*, pages 465–479, Berlin, Heidelberg, 2008. Springer.

[30] D. L. Moody. Why a diagram is only sometimes worth a thousand words: An analysis of the bpmn 2.0 visual notation. 2011.

APPENDIX A CONCEPTUAL MODELS

This Appendix contains conceptual models that were created from the developed notation. These models show some of the ways that the notation may be used to visualize mobile systems, or parts of them.

A.1 Carbon Track and Trace models

This section contains conceptual models that capture some aspects of the information systems used in the Carbon Track and Trace project[2][3]. Chapter 4 discussed the basic architecture of the CTT project, while Figure 4.4 showed its architecture and the flow of data between its components. Figure A.1 is a model that attempts to represent this architecture with the modeling notation as well as additional information that could be useful to represent (the specific values used in this model are not accurate to reality, but rather shows an example realization of the architecture within the notation). This section will divide Figure A.1 into a number of smaller parts and explain them.

Sensing data is observed at two locations in the model, shown in Figure A.2a and A.2b. Both locations have a sensing unit that sends the sensor data to a gateway. Both sensing units have some sensors inside them, and the unit in Figure A.2b also has a visual BPMN process in it which shows that the unit recalculates its sending interval based on feedback that it receives. The BPMN part of this figure is shown in a close-up in Figure A.3. All the data from the sensors is aggregated at a research lab and the data is stored on a cloud storage. Figure A.4 shows the data being stored at the research lab, and how it is composed of different kinds of data such as traffic, air quality and emission data. This is the stored data which is used for further analysis. This data is also used by the Dataport system which creates network visualization from the data.

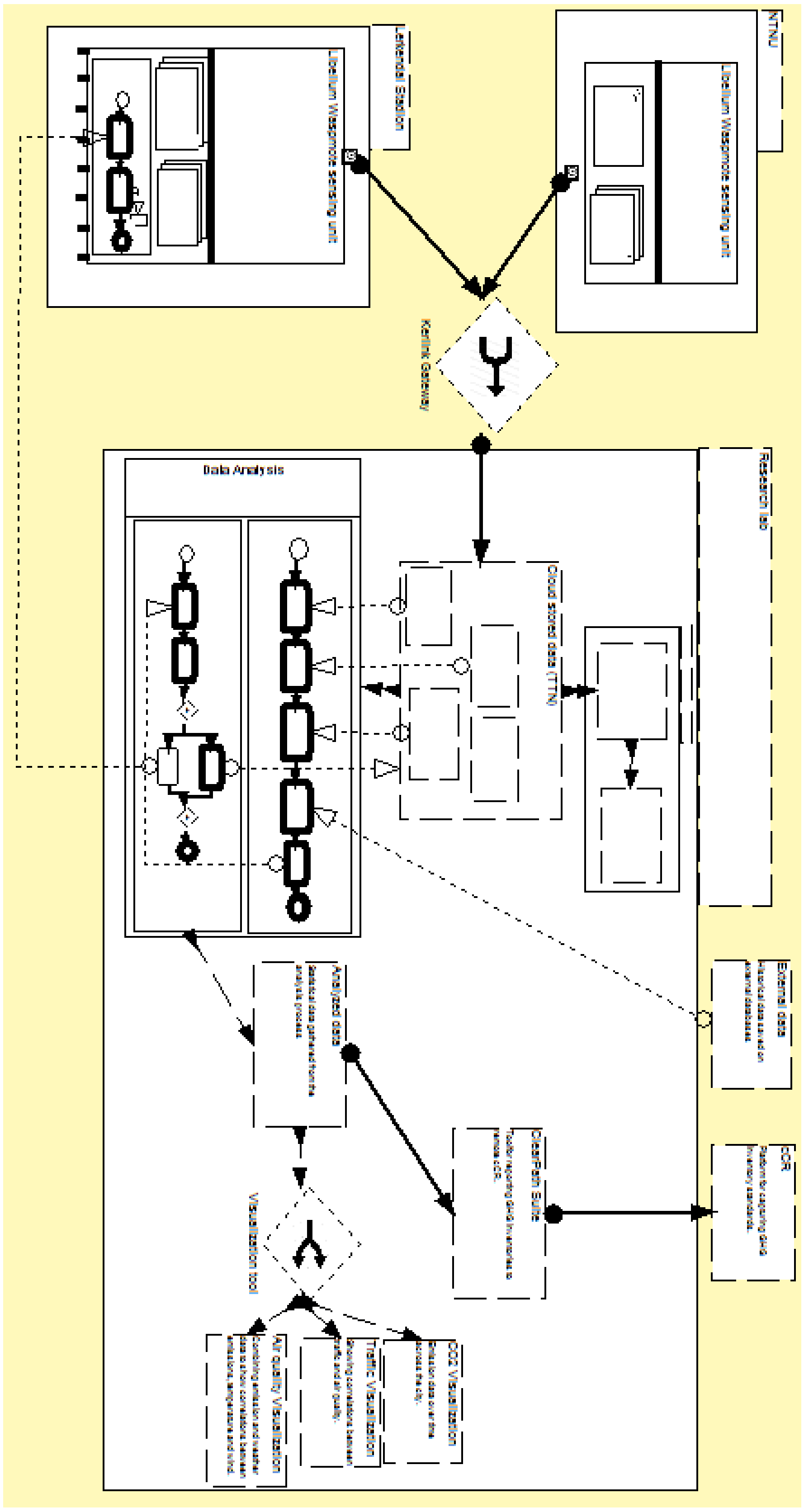
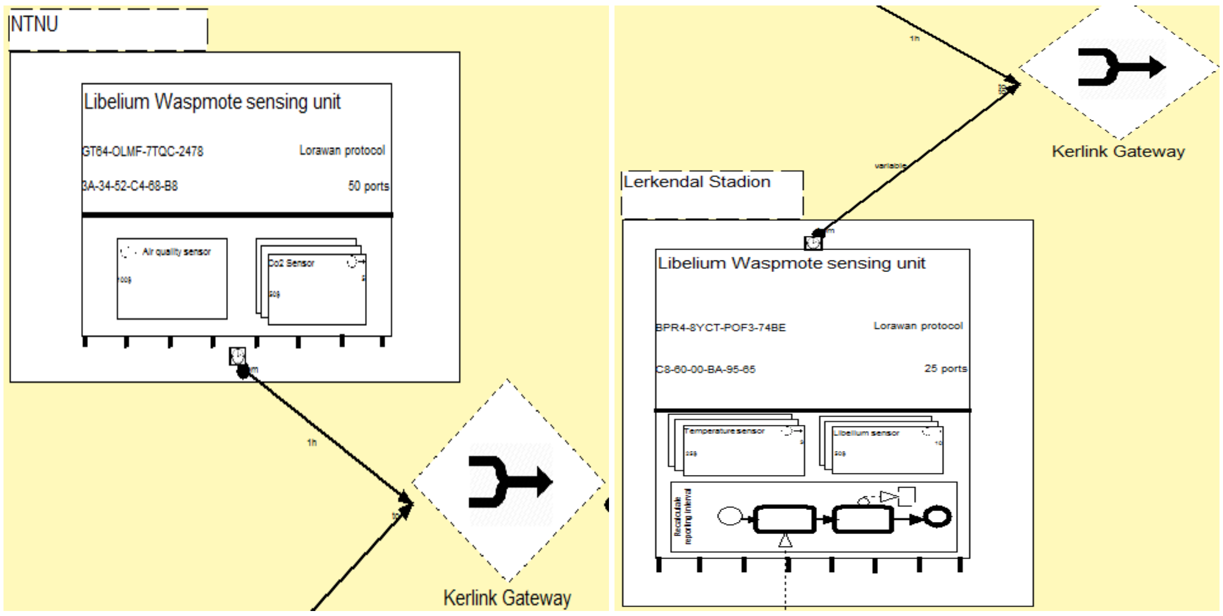


Figure A.1: Conceptual model of the basic CTT architecture.



(a) A sensing unit (b) Another sensing unit with a BPMN process

Figure A.2: The two sensing units and the gateway used in the model

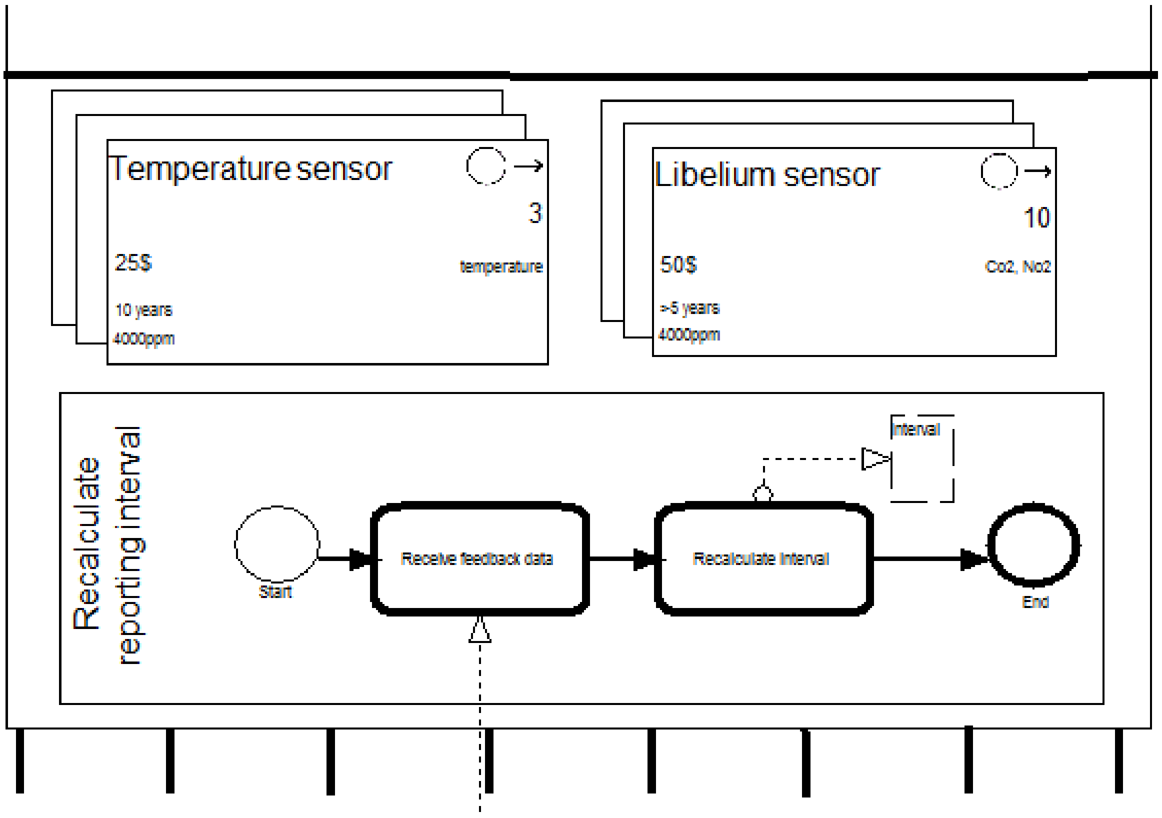


Figure A.3: Closeup of the BPMN process in Figure A.2b

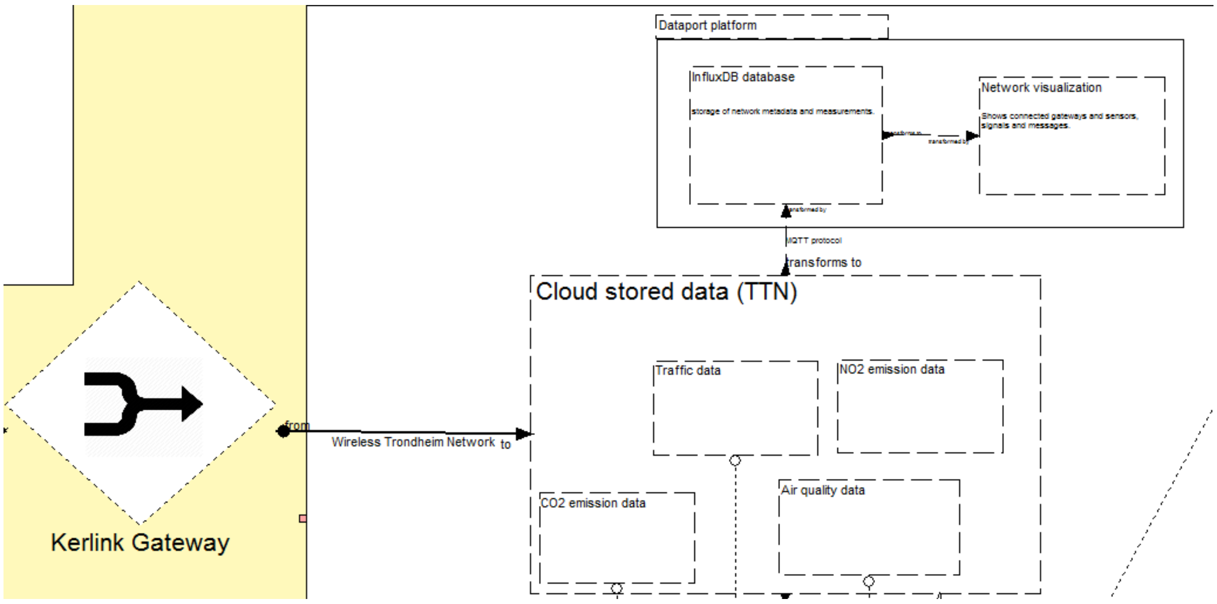


Figure A.4: Data being stored in cloud storage

Figure A.5 shows a closeup of the BPMN diagram in Figure A.1. The top process is the data extraction where the different kinds of data in cloud storage is fetched and compiled. The compiled data is then used in the analytics process where the data is analyzed, then saved back to the database and used for feedback to the sensing units. Figure A.1 shows the data going from this process to the sensing unit in Figure A.2b. The final analyzed data is also used to create visualizations of the data, which is shown in Figure A.6.

The different visualizations in Figure A.6 are useful to different end users. One end user of interest is the cCR registry of GHG emissions. The ClearPath Suite in Figure A.6 is used to report the data to the external cCR registry, which is shown in Figure A.7. As seen in this Figure, the cCR registry and the external data used in the analysis process have been put outside of the research lab location, because they are external factors outside of this location.

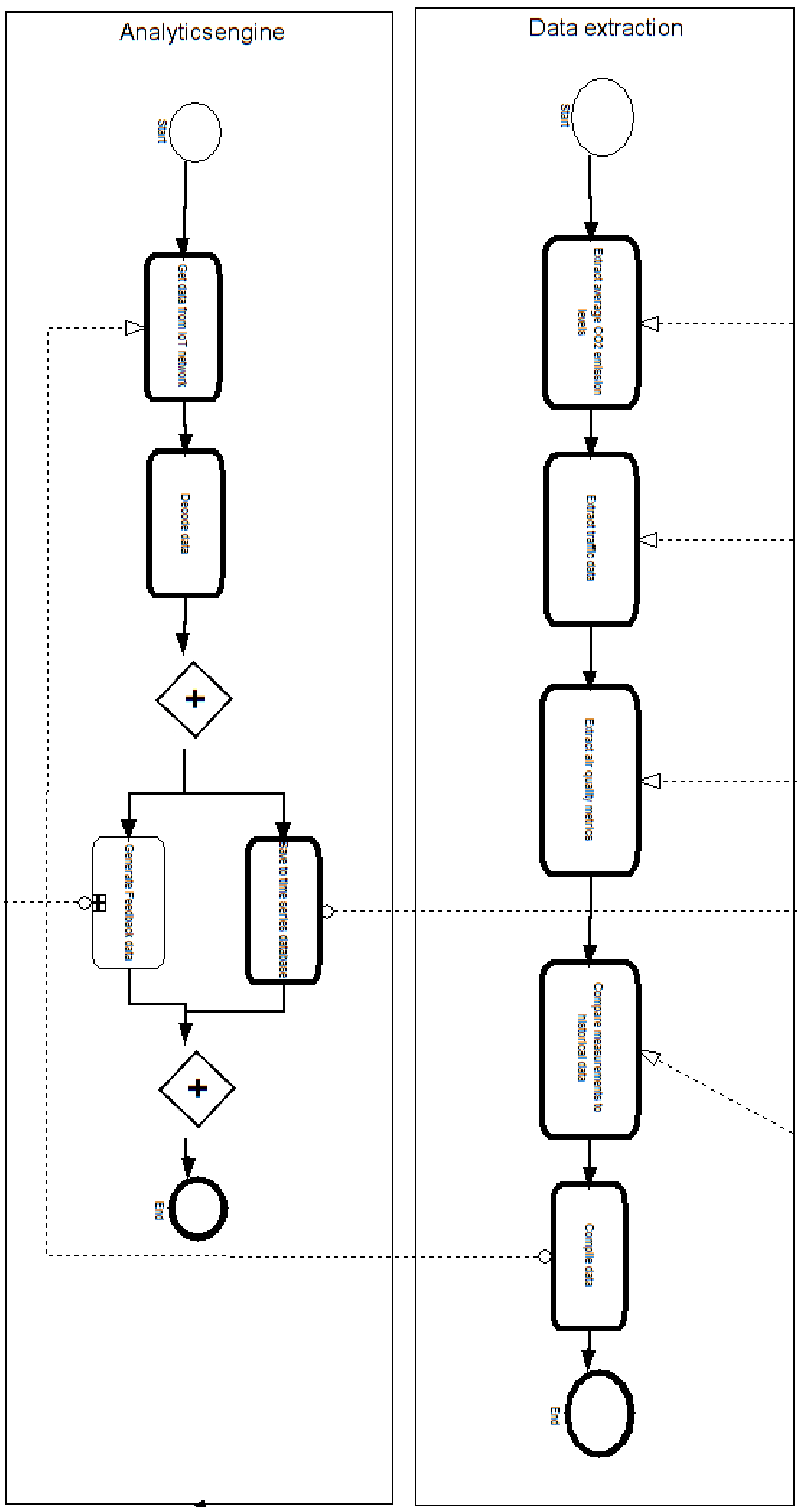

Analyticsengine
Start
Get data from IoT network
Decode data
Generate Feedback data
Save to time series database
End
Data extraction
Start
Extract average CO2 emission levels
Extract traffic data
Extract air quality metrics
Compare measurements to historical data
Compile data
End

Figure A.5: BPMN process of extracting and analyzing sensor data

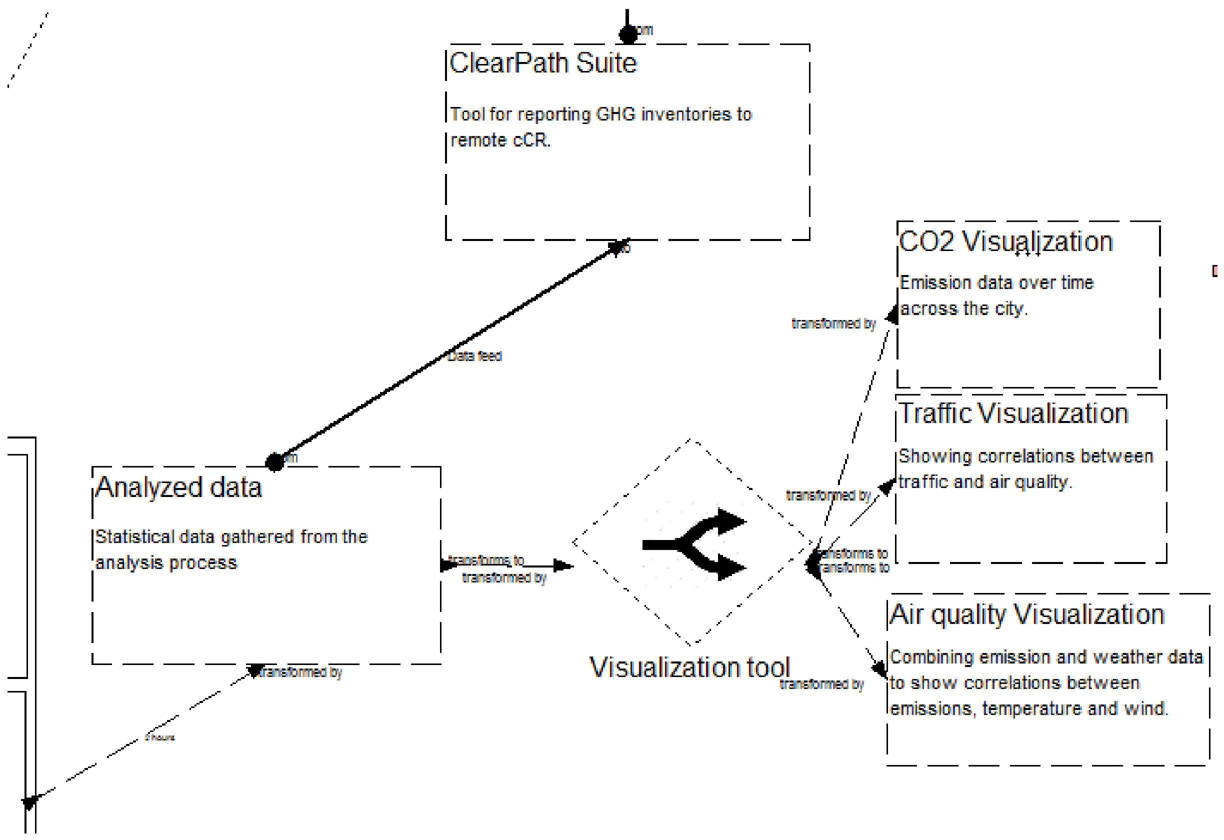

ClearPath Suite
Tool for reporting GHG inventories to remote cCR.
from
to
Data feed
CO2 Visualization
Emission data over time across the city.
transformed by
Traffic Visualization
Showing correlations between traffic and air quality.
transformed by
Analyzed data
Statistical data gathered from the analysis process
from
transforms to
transformed by
Visualization tool
transforms to
transforms to
Air quality Visualization
Combining emission and weather data to show correlations between emissions, temperature and wind.
transformed by
transformed by
3 hours

Figure A.6: Analyzed data being applied in visualizations

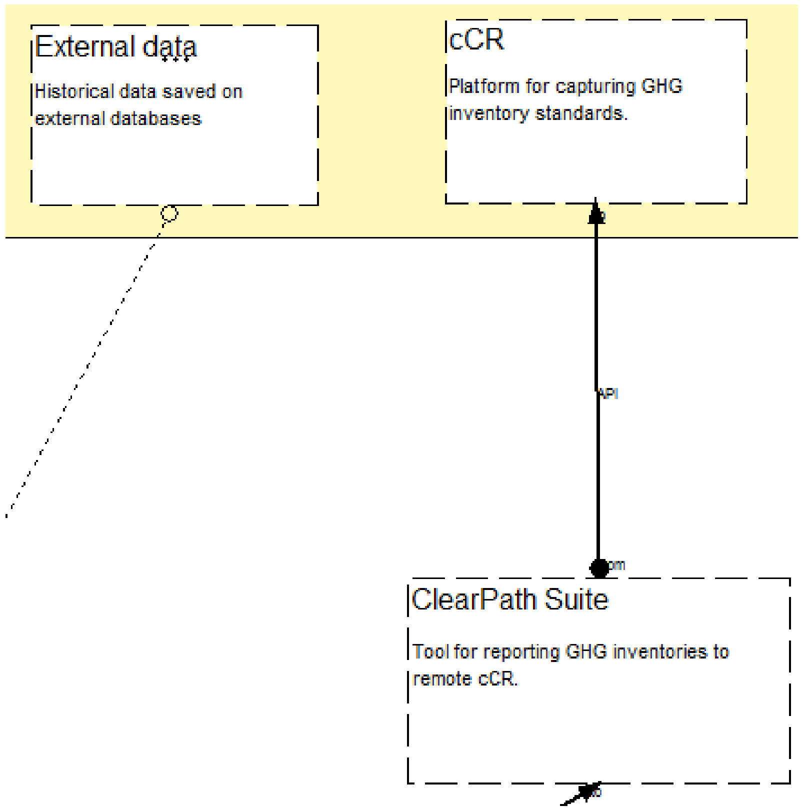

External data
Historical data saved on external databases
cCR
Platform for capturing GHG inventory standards.
to
API
from
ClearPath Suite
Tool for reporting GHG inventories to remote cCR.
to

Figure A.7: External data

APPENDIX B INSTALLATION INSTRUCTIONS

This section details how to access the metamodel described in the thesis, using the Metis tool.

Needed files:

- Metis client (version 5.2.2 used for creating the notation)
- The metamodel itself (folder named Xml.master)

Steps:

1. Copy the http folder from the metamodel into Metis http folder (default location: C:\Program Files (x86)\Metis\Metis5.2\xml\http)
2. Copy the startup folder from the metamodel into Metis startup folder (default location: C:\Program Files (x86)\Metis\Metis5.2\xml\startup)
3. Launch Metis in local license mode. the "mobile project modeling" metamodel should be automatically loaded in the "metamodel" tab. This metamodel contains all the objects used for modeling with the notation.
   - If the notation does not load automatically, it can be opened manually from *File -> Open* within Metis. The metamodels can be found in the "metamodels" folder within xml.master.
4. To create models with the mobile notation, select *File -> New -> Model*, and select the *Project Blank Mobile* template. This will create an empty model with all the mobile notation constructs available.
5. For editing the notation, open the metamodel named "mobile". This metamodel contains all the objects from the "mobile project modeling" metamodel, as well as all the typeviews used by these objects.