Vrije Universiteit Brussel

FACULTY OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

# A Framework to Provide User Control in Context-aware Systems

Graduation thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

Wouter Mensels

Promoter: Prof. Dr. Beat Signer
Advisors: Sandra Trullemans


Academic year 2013-2014

Vrije Universiteit Brussel

FACULTEIT WETENSCHAPPEN
VAKGROEP COMPUTERWETENSCHAPPEN

# A Framework to Provide User Control in Context-aware Systems

Afstudeer eindwerk ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad Master of Science in de Ingenieurswetenschappen: Computerwetenschappen.

Wouter Mensels

Promoter: Prof. Dr. Beat Signer
Advisor: Sandra Trullemans


Academiejaar 2013-2014

# Abstract

Context-aware applications modify their behaviour based on the current context. Information about this context does not have to be explicitly provided by users. Part of this information can be automatically detected or derived by the application. In its most simple form, contextual information can be directly sensed by physical sensors (e.g. location and temperature). On the other hand, more complex human-related context information such as user tasks, goals or preferences needs to be derived by applying high-level reasoning. Given the variable nature of human behaviour, these human-related contexts cannot be deterministically modelled upfront. Context-aware applications that exploit human context need to involve end users when processing context. End users need to have an understanding of how the system detects context and need to be able to control this process.

In this thesis, we present a framework that enables end users to define their own contexts by creating models. Context is considered as generic as possible where users should be able to define every context they need. In order to provide generic context models, the framework needs to be flexible in easily integrating new types of context information. Since end users usually do not have programming skills, they require straightforward user interfaces that are easy to use. These tools typically have limited expressive power. Therefore, we present a context modelling framework which combines flexibility and extensibility with easy to use modelling tools that can be used by end users. The framework consists of three layers namely a plug-in layer, a configuration layer and a context layer. Each layer covers a part of the modelling tasks, produces different deliverables and is targeted to users with different profiles. Deliverables created in one layer are used in the next layer. Next to the modelling tools, the framework provides a runtime environment for the real-time evaluation of user-defined contexts. This runtime environment uses a rule engine. The framework translates user-defined context models in declarative rules that can be loaded in the rule engine.

As a proof of concept we have applied the context modelling framework to the domain of *Personal Information Management* (PIM). It is natural for humans to use context when organising and re-finding personal information items. This implies that context-aware computing can make a significant contribution to PIM systems. We used the modelling tools to create a user-defined context and integrated the runtime environment of the framework with a context-aware desktop application.

# Acknowledgements

First of all I would like to thank my advisors Brecht De Rooms and Sandra Trullemans for their support, frequent reviews and feedback. Even for me as a working student with several years of professional experience and a previous Master's Thesis you have managed to set new standards for quality that I have acquired for live.

I would further like to thank the teaching staff at the Computer Science Department of VUB and especially the WISE research group for their flexibility towards working students. At several occasions additional courses and explanations were provided after business hours, often to a very small group of students. The combination of study and work is a balanced exercise for which all assistance and flexibility is highly appreciated. And of course I want to thank my promoter for his advice and assistance.

Many thanks also to my friends and family, and especially to my partner Heidi and her children Luckas, Marie and Katoo. I realise that living with a working student can be demanding, especially during exam periods. Thanks a lot for your patience and support!

# Contents

# 1 Introduction

## 1.1 The Ubiquitous Computing Paradigm

During the 1980s, personal computers became available to a broad public at affordable prices. In 1993 more than 150 million computers were in use worldwide[1]. It seamed that computers started to become part of our daily lives. However, these computers were not integrated with their environment. They were installed at a fixed location as stand-alone devices. Personal computers were only capable of responding to input explicitly provided by their users, in a strict format dictated by the computer application. These shortcomings were addressed in 1991 by Mark Weiser in his paper 'The Computer for the 21st Century' [65]:

> *"More than 50 million personal computers have been sold, and the computer nonetheless remains largely in a world of its own. It is approachable only through complex jargon that has nothing to do with the tasks for which people use computers"*

In this paper he introduced his vision of *ubiquitous computing*. This vision refers to a world where people are surrounded by computing devices without being aware of it. These devices would provide functions to assist users in

[1] source: ITU

Figure 1.1: Various sensor components from the Arduino product family, many sensors cost less than one Euro.

their everyday tasks and would do so in an unobtrusive manner. People would use these functions unconsciously without even noticing the devices that provide them.

Twenty years later, we can see that this vision has been partly realised. Advances in hardware, network and sensor technology have provided low cost components that support new ways of interaction. An example are accelerometers in portable devices that derive their position based on sensed gravity. Computing devices have taken different shapes. Many of the devices predicted by Weiser have been realised in the form of smart phones or tablets. People started to use computer technology everywhere and at any time. At the same moment, interfaces have become more intuitive. For example, many applications developed for children can typically be used without any explanation. While we can agree that we are indeed more surrounded with computing devices, these devices have not become unobtrusive. They are not capable of anticipating our needs and provide us with the appropriate service at the appropriate time without being explicitly instructed to do so. Attempts to do so have often resulted in applications that are more a hindrance then a help since they frequently made incorrect assumptions regarding user needs. As a result users abandoned them. The *calming* or *invisible* aspect of the ubiquitous computing vision remains difficult [34].

With the introduction of ubiquitous computing, *context-aware computing* became an important research field. Context-aware computing concerns applications that use information derived from context. Whereas traditional applications rely on explicit user input provided through mouse and key-


Figure 1.2: The Tripadvisor app uses location as context information. http://tripadvisor.en.softonic.com/

board, context-aware applications have access to other information sources such as sensors. Dey and Abowd [2] argue that context-aware computing improves the human-computer interaction:

> *"By improving the computer's access to context, we increase the richness of communication in human-computer interaction and make it possible to produce more useful computational services"*

An example of a context-aware application is TripAdvisor [2], a mobile app for tourists that presents relevant information (e.g. nearby restaurants, hotels and places of interest) to a user depending on his current location. Figure 1.2 shows information as presented by the app.

Context information can take many forms, varying from simple data like the current location, surrounding noise levels and temperature to more complex information such as a users general preference. Applications that derive contextual information from sensors are widely available. However, context is not restricted to data that can directly be measured through a physical sensor. From a human perspective, context can also relate to information such as a users tasks or goals. Hofer [38] makes the distinction between *physical* and *logical* context to differentiate between context data that can be sensed from the physical environment and the more complex context that humans can define. It is much harder to exploit logical context since it cannot directly be

[2] http://tripadvisor.en.softonic.com/

measured. Therefore, logical context requires complex context models and reasoning. Moreover, it requires the fusion of data acquired through various sources such as the combination of data gathered from sensors with more static data from databases. An example of a context-aware application that exploits logical context is an online chatting tool that presents the availability status of a user (e.g. available, busy, in meeting or do not disturb). An availability status can be derived from context information such as a user's calendar, the time of day and personal preferences.

One might assume that the exploitation of logical context is just a more complicated extension of the exploitation of physical context. However, there appears to be a structural difference between physical and logical context. Several researchers argue that it is impossible to deterministically model complex human-related context in advance, as is usually done for physical context. Belotti and Edwards [9] point out that any attempt to do so will fail since there is too much variability in human behaviour. They propose a new approach where users are involved in the processing of complex human-related context. Users should have a good understanding on how a context-aware application operates on context and should be able to influence this process.

## 1.2 Contribution of this Thesis

In this thesis we address the issue of context modelling. We consider context to be as generic as possible: every possible combination of context information items that a system has access to is a possible context. This includes complex human-related contexts for which end users need to be involved. To achieve this we provide end users with the means to create their own context models. Since end users typically do not have programming skills they require straightforward and easy to use tools. At the same time we want to maximise the expressibility of these tools. The system should also be extensible to allow an easy integration of new sources of context information.

To combine the requirements of end user programming, expressibility and extensibility we have created a context modelling framework that consists of three layers: a plug-in layer to support extensibility, a configuration layer to maximise expressibility and a context layer to provide an easy to use modelling interface for end users. These layers each cover their own part of the modelling activities and are targeted to users with a different profile: plug-in developers, expert users and end users.

**Plug-in Layer:** Since one framework typically does not fit all domains, we provide a framework that allows programmers to create packages for a

specific domain which can be used in the other layers. Packages for different domains can be combined in a flexible way.

**Configuration Layer:** In the second layer, an expert user creates templates which are the basic building block for the end user. For the creation of templates we provide the *ExpertUser Tool*, a tool with a graphical interface. Creating templates requires no programming.

**Context Layer:** In the third layer, an end user combines the context information items using the templates created in the second step. For this task we have created another tool with a graphical interface: the *EndUser Tool*. The context layer supports the creation of user defined events, which essentially allows the user to give his own interpretation to any combination of context information items.

To our knowledge, this is the first framework that focuses both on low-level data delivery as well as on the possibility to let end users understand and reprogram the system through visual tools without a restriction to a certain domain. Since we can deliver new packages with conditions and create new templates in the ExpertUser Tool we do not focus on one particular scenario or domain. We believe that the separation in three layers, the separation of users in three categories, the two visuals tools and the ability for end users to define their own events provides our framework with highly desired features such as extensibility, flexibility and end user friendliness without sacrificing expressiveness.

Next to the tools that support the modelling activities, we have integrated a runtime environment into the framework. This runtime environment consists of a rule engine. The framework translates user defined contexts in declarative specifications that can be loaded in this rule engine. Plug-ins also have runtime components that collect context information and forward this information to the rule engine that performs a real time evaluation of user defined contexts.

As a proof of concept we have applied our framework to the domain of personal information management (PIM). As argued by Trullemans [56], context can facilitate the retrieval of previously stored information items. She created the *Context-Aware Desktop*, an application that presents users a different virtual desktop depending on the current context. We have integrated this application with the runtime environment of our framework. We added a user-defined context and simulated the basic context information items required to activate the user-defined context in the *Context-Aware Desktop*.

## 1.3 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 introduces context and context-aware computing. We explain context from a general perspective and from an engineering perspective. We continue with an overview of context-aware computing and address the problems of context modelling. Next we explain why the user needs to be involved in the creation of complex context models, introducing the concepts of *intelligibility* and *control* for context-aware applications. Chapter 3 elaborates on the architecture of context-aware computing further. We present some generally accepted architectural design principles, and investigate existing frameworks for the creation of context-aware applications. Chapter 4 introduces our framework for end user context modelling. The general structure of the framework and its different modules are presented. We discuss the plug-in layer, configuration layer and the context layer. This chapter focuses on the capabilities of the framework, and how to use the tools. Chapter 5 gives the implementation details of the components that constitute the framework. It explains what libraries are used and how the modules are structured from a programming point of view. Chapter 6 presents a use case related to Personal Information Management (PIM). We give an introduction to PIM and explain why context is important. We then use the modelling tools to create a user-defined context related to the PIM domain. The context-aware desktop application is integrated with the runtime environment of the framework. Chapter 7 presents our conclusions and suggestions for future work.

# 2

# A User-centric Approach to Context

This chapter explains how context information can be used in computer applications. We start with an introduction of the concept *context* where we provide definitions both from a general and from an engineering perspective. Next we discuss some of the issues that context-aware computing has introduced. We explain why end users need to be involved when applications process context, and finally we formulate the requirements for a user-centric context modelling framework.

## 2.1 Introduction to Context

The concept of context is studied in multiple domains, ranging from philosophy, cognitive psychology, pragmatics and linguistics to human computer interaction and artificial intelligence. First, we introduce context from a general perspective. Second, we investigate context from an engineering perspective with an introduction to context-aware computing. We provide several definitions within these perspectives and explain how they differ and overlap.

### 2.1.1 Context from a General Perspective

In the late eighties, the concept of context has taken a central role in various disciplines concerning knowledge representation [11]. These cover engineering disciplines as well as domains more positioned in human sciences. Researchers have reached the common understanding that organisms, objects and events are integral parts of the environment [26]. However, different research areas each have their own approach and definitions of the concept *context*. Benerecetti, Bouquet and Ghidini [11] indicate that a general and unifying theory or formalism of context has not yet been established. They observed the fact that we are even not sure whether each research area is handling aspects of the same problem when applying context. Dervin [27] pointed out that although context is frequently cited in multiple areas, the overall concept of context itself has not received much philosophical and theoretical treatment. She describes two extreme views on context. The first view defines context as any possible analytic factor of the phenomenon under investigation. This allows virtually every possible attribute of person, culture, situation, behaviour, organisation or structure to be context. Being a clearly defined and separated attribute, the impact of context on a phenomenon under investigation can be analysed in the traditional scientific sense. For example, for a researcher focusing on human behaviour, attributes related to income, age and education can be context. In the second view, context cannot be resolved into separate factors. It is the general surround of a phenomenon without which any possible understanding of human behaviour becomes impossible. In this view every context is different and an analytical approach is not possible. Dervin proposes a definition of context that takes a position in between these two views :

> *"Context constitutes the necessary conditions for sufficient understanding of phenomena."*

### 2.1.2 Context from a Computing Perspective

The term context-aware was introduced by Schilit and Theimer [50] in 1994. As part of the ubiquitous computing paradigm introduced by Weiser [65], context-aware computing has become an important research field. The integration of context in ubiquitous computing is inspired by the important role that context plays in human-to-human communication [24]. Dey et al. [2] pointed out how successful humans are at conveying ideas and reacting appropriately. This is not only due to the expressiveness of the used language but also to a common understanding of everyday situations. Humans are able to use this implicit situational information to increase the conversational

bandwidth. If computers would have access to this implicit (i.e. context) information, human-computer communication could be enriched. This would lead to more useful computational services.

It is generally accepted that the 'Active Badge Location System' presented in 1992 by Want et al. [63] was the first context-aware application. It used infra-red technology to locate a user in a building and used this information to automatically forward phone calls to a telephone nearby the user. The following years, several context-aware tour-guides were developed [54, 1]. These tour guides present information to a user based on the user's current location.

In the context-aware computing research community, several researchers have proposed their definition of the concept context. An overview of these definitions is provided by Dey and Abowd [2]. They illustrate how some of the early definitions try to define context by enumerating a list of examples [18, 51]. These enumerations typically consist of elements such as location, time, objects and people nearby. Dey [28] extends this list with elements that apply to more human related aspects of context such as emotional state, focus of attention etc. Other definitions try to define the concept of context by providing synonyms like environment [17], situation [35], or state [64]. However, Dey and Abowd claimed that these definitions are difficult to apply in practice. Defining a concept as an enumeration or by providing synonyms does not help when trying to identify what data can be considered as context. Schilit et al. [50] provided a more operational definition where context is defined as the constantly changing execution environment. The environment in this definition consists of three parts:

- *Computing environment:* available processors, devices accessible for user input and display, network capacity, connectivity, and costs of computing.
- *User environment:* location, collection of nearby people, and social situation.
- *Physical environment:* lighting and noise level.

The definition of context given by Dey and Abowd [2] is probably the most accepted in the pervasive computing community, due to its general nature:

> *"Context is any information that can be used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves."*

The definition by Dervin [27] given earlier specified context to be the necessary conditions for sufficient understanding of phenomena. If we compare these two definitions, we see that both define the context of a concept (this concept being for example an entity or a task) to be additional information surrounding this concept. The definition of Dey however restricts context information to items related to an application. Since we are interested in every possible user context, we will follow the more generic definition given by Dervin in this thesis.

Henricksen [37] points out that there is still a lack of precision and consensus in definitions of context provided by the ubiquitous computing community. She explains this by the lack of separation between *context*, *context models*, and *context information*:

- *Context of a task:* The circumstances surrounding a task that are relevant to its completion.
- *Context model:* A subset of context that can realistically be sensed or derived by an application, and can be exploited in the execution of a task. It is explicitly specified by an application developer in a formal language.
- *Context information:* A set of data according to the context model.

We illustrate these three concepts with the example of a museum tour guide application. In this example the context of the visiting task is the general situation of a visitor while visiting the museum. The context model can be restricted to the position of the visitor and objects nearby that position. Context information consists of the actual values measured during the visit. Henricksen points out that since applications work with well defined context models, definitions for the context of tasks are less interesting for context-aware applications. This thesis will focus on the context modelling task. However, unlike Henricksen we do not consider this task to be the sole responsibility of the system designer.

An important segmentation that can be applied to context information is the one introduced by Hofer et al. [38]. They make a differentiation between physical and logical context. Physical context refers to information that can be measured by sensors (e.g. light, sound, movement, temperature) whereas logical context refers to the human aspect of context. Logical context can involve user goals, tasks or emotional state. Furthermore, it has either to be explicitly specified by a user or must be derived from information captured by monitoring user interactions. Location is the earliest type of physical context that was actually used in context-aware applications [63]. Examples of context-aware applications that use logical context are the Watson

project [19] and the Intellizap project [33]. Both use previously accessed information items as context information. The following section elaborates on issues encountered when working with logical context.

## 2.2 A User-centric Approach

During the last two decades, several context-aware applications and frameworks have been presented. Context-aware computing has however introduced new concerns. Since context-aware applications operate on implicit situational information they typically require less explicit user input. As a result, users can loose control over the behaviour of the application. Some of these issues are introduced by Bellotti and Sellen [10] in their research regarding the design of context-aware computing environments. The authors argue that humans have natural mechanisms of feedback and control when interacting with other humans in order to protect them from undesired sharing of information. When systems take over some of the functions that mediate between people, it is possible that these mechanisms of control and feedback are lost. Users have often no adequate feedback over what information they expose to a system and they are not able to control the processing of this information. For example, a context-aware application that uses location of users might actually expose a users location to other users. Users can feel uncomfortable with the idea that information regarding their location is being shared.

Bellotti and Sellen argue that technology results in *disembodiment* and *dissociation*. In human-to-human communication, people transfer cues on top of the explicit verbal communication channel (e.g. non-verbal communication). In machine to human communication, this information stream is *disembodied*. It is reduced to the formal vocabulary that a system supports, additional contextual information (e.g.that the information is confidential) is easily lost. *Dissociation* means that the result of an action is visible but the action itself is not. A result of dissociation is that a user cannot respond to an action or request since the user is not aware that the action was triggered. In order to remedy the issues of disembodiment and dissociation, Bellotti and Sellen present a conceptual framework that addresses the design of *control* and *feedback* in context-aware systems. Control refers to the possibility for end users to stipulate what information they expose to a system, and who has access to this information. Feedback means that people are informed about the kind of information that is being captured about them, when it is captured, and to who this information is made available. In their framework, the authors state that *feedback* and *control* can be applied to the *capture*,

*construction*, *accessibility* and *purpose* of information:

- **Capture:** Applies to the kind of information that is being picked up (e.g speech, image, etc.), including the type of sensors used for gathering info, their location, etc.
- **Construction:** The processing and storage of information after capturing. Information can be stored or not, can be encrypted or not, etc.
- **Accessibility:** Information can be publicly available or availability can be restricted to a limited set of users, etc.
- **Purpose:** Applies to the intentions people might have when using information.

These four classes of concerns are not independent. Feedback over the capture of information is most important since it enables users to adapt their behaviour. For example, users can modify their behaviour if they are aware that they are in an area where a camera is active. Accessibility of information also influences the capture of information: users might behave different near a camera depending on who has access to the information captured by the camera. Based on this framework, several criteria are proposed to assess system design with regards to feedback and control. These criteria indicate that feedback should be noticeable but in an unobtrusive way. This implies that feedback should be selective and relevant for a user at the time it is given.

Besides the importance of feedback and control to mitigate privacy related issues, a more structural issue is identified by Bellotti and Edwards [9] regarding the modelling and processing of context information. For existing context-aware applications that work with physical context, the context model is typically defined by the application designer. This context model is fixed at design time with little or no possibility for the user to create or modify it. Users can at best specify preferences to customize the behaviour of their context-aware application (e.g. specify if calls need to be blocked when the user is a meeting). Users cannot define for themselves what attributes are relevant in a context model (e.g. specify the context information items that the system uses to identify the 'in a meeting' context). Bellotti and Edwards have pointed out that this approach does not work for applications that make use of human context [9]. When human behaviour form part of the context description, it becomes impossible to model every aspect of context at design time. There is too much variability in actions a system should take

for designers to model all outcomes in advance. Unlike systems, people make unpredictable and non-deterministic judgements about context. Bellotti and Edwards argue that end users need to be involved. End users need to be able to control how the system uses context. In order to control the behaviour of context-aware systems, users need to understand what the system is doing. To formalize the involvement of end-users in the behaviour of context-aware systems, Bellotti and Edwards introduce the concepts *intelligibility* and *accountability*. A context-aware system is *intelligible* to the user if this user has insights in its processing. This implies that users need to know what context the system has identified, how it identified this context and what it will do with this information. Next to providing intelligibility, context-aware systems must enforce user *accountability*. This means that a user must be made accountable for actions the system takes on his behalf. Accountability is important in situations where computers start to mediate between people. People are uncomfortable when confronted with actions taken by an anonymous system. As mentioned earlier in this section, Bellotti and Sellen refer to this issue as the problem of *disembodiment* [10]. To support intelligibility and accountability in context-aware applications, Bellotti and Edwards [9] propose that systems should inform users about their capabilities. Systems should further provide feedback about the capture, construction and accessibility of information, and about its the purpose. And finally, systems should provide the appropriate level of user control. A balance needs to be found between minimizing human effort on one side and providing the desired outcome at the other side. The level of control a context-aware system needs to provide depends on how certain the system is about the desired outcome based on the current context. In case the desired outcome can be derived with high degree of certainty, the system can initiate the action without bothering the user. It is sufficient that the user has ways to correct the action a system takes. In case there is more doubt regarding the desired outcome, the system needs to ask confirmation before proceeding. When there is too much doubt to select any action at all, the system needs to present the user a set of choices before taking any action. With regards to the modelling of context, Bellotti and Edwards [9] propose to use rich, ambiguous models that do not fully specify the context, but leave room for interpretation by the end user. Instead of allowing machines to interpret all context information, users should be informed and allowed to complete or modify the detected context.

To investigate users' sense of control in interactive applications, Barkhuus and Dey [7] have performed a case study. In this study, the authors assessed participants' reactions and attitudes towards context-aware applications and interactive applications in general. Three types of interactivity are given:

personalisation, active context-awareness and passive context-awareness. Interactivity through personalisation means that a system does not take initiatives to change its behaviour. All initiative comes from the user who is allowed to change the system behaviour. This can typically be accomplished by changing settings such as setting ringtones or background pictures on a mobile phone. Active context-awareness applies to applications that change their behaviour autonomously based on sensed or derived context. They do not ask confirmation. An example of active context-awareness is a mobile phone that automatically changes its time settings when it detects entering a different time zone. Finally, passive context-awareness applies to applications that sense or derive a context that they present to the user. They do not autonomously change their behaviour. When presented with the context, the user has the possibility to change the application behaviour. An example of passive context-awareness is a mobile phone that informs the user when it detects entering a different time zone. The application does not autonomously change the phone's time settings, it only suggests the user to do so.

The study shows that people feel less in control when using context-aware services than using applications which only allow personalisation. Although participants still prefer active and passive context-aware services over personalisation, the study illustrates that participants might become frustrated by the perceived lack of control in context-aware services. Users are willing to accept a certain level of autonomy only if the application's usefulness justifies the associated loss of control. Applications that autonomously derive human aspects of context will however always be error prone [9].

To improve the intelligibility of context-aware systems, a lot of research has been performed on how applications can generate explanations for end users that allow them to understand how the application processes context [43, 44]. Lim and Dey [45] agree that users have a need for intelligibility to improve user satisfaction, adoption and acceptance. However, they also show that users may not be interested in all system generated information. Generated explanations should adapt to the varying situations of use. In a study about requirements for intelligibility, Lim and Dey [45] assess what types of information users are interesting in, and how intelligibility improves user satisfaction. The study shows that users are interested in different types of information depending on the type of *application* and the current *situation*. In general, information about the application is highly desirable. People are especially interested in the application model, which specifies the behaviour of the application. Information about the detected situation was moderately important. Knowing how to control an application was also an important

requirement. People have more demand for intelligibility types when they are aware that such types are available, or when the situation is more critical. Furthermore, Lim and Dey acknowledge that other explanation types (e.g. history) that could be considered important, were not addressed in the study. Though the study provides useful information on how to provide intelligibility, it only applies to the content of information. The authors address the question of what information is required to support intelligibility and control. However, next to the content of information, intelligibility and control are also influenced by how the information is presented. Vermeulen [62] indicates that there are different ways of presenting information to support intelligibility. Designers have to decide when to present information related to events (e.g. before, during or after the event takes place), who takes the initiative for showing the information (the user or the system), and what kind of control is provided to act upon the information. Furthermore, different types of interfaces can be used. Multiple modalities can be used and different levels of integration are possible between the interfaces and the rest of the application.

Lim and Dey [47] have presented a formative study on how to present and provide explanations to help users understand and trust the autonomous behaviour of context-aware applications. They have developed LaKsa, a context-aware application for mobile devices that shares people's availability status. The authors used this application to explore the design, implementation and use of intelligibility. From this study, the authors propose four design recommendations on how to present information to support intelligibility:

- Reducing and aggregation explanations: information needs to be presented at an aggregated level, with the possibility to demand more detail if needed.
- Explanations should be provided using simple and information specific components: Simple components that answer all questions on a related part of context (e.g. regarding location) are better than providing a separate component per question that covers multiple parts of context.
- Streamlining questions: Intelligibility questions are not independent, one question usually leads to a next question.
- Real-world explanations: In order to fully understand a context-aware system, a detailed knowledge on the application domain is needed. To integrate intelligibility with the application domain, systems might need access to complex real-world concepts that might originally be out of scope of the application.

Further research on the presentation of intelligibility was performed by Vermeulen [61] who has extended the ReWiRe framework with support for intelligibility. Vermeulen has extended a rule based context-aware system with additional meta-data that links the rules together. By processing these annotations at runtime, a model of the system's behaviour is created. Based on these extensions, the PervasiveCrystal system is created which answers *why* an *why not* questions [62].

Next to the complexity regarding intelligibility, a similar complexity exists regarding control. The importance of user control in context-aware computing is pointed out by Van der Heijden [60], who argues that the transfer of control from users to applications results in an increase of user discomfort. From these arguments one can conclude that context-aware applications need to focus on user control. However, increasing user control means reducing the autonomy of a context-aware application. Since autonomy is one of the key aspects of context-aware computing, it cannot be abandoned: applications need to find a correct balance between autonomy and user control. Hardian, Indulska and Henricksen have performed a survey on this topic [36] where they find that the issue has received little attention so far:

> *"The challenge of designing applications to provide appropriate control to users has traditionally taken a back seat to more fundamental problems in context-aware systems, like sensing and interpreting context."*

Context models have been considered for internal use by the application only, not to be exposed to users for understanding or control. One of the exceptions is the Jigsaw editor [39] that supports end-user configuration.

## 2.3 Requirements for a Context Definition Framework

For an application to successfully make use of context, it must sense the environment and recognise information that can be relevant for a user [15]. For the first task, various cheap sensors exist. Some have already been commercially launched (e.g. Microsoft Kinect). For the second task, the context model is of critical importance since it defines what context information the application can benefit from. Many techniques exist for modelling context, varying in expressive power and support for reasoning [13]. This thesis will focus on the development of a framework for context modelling where context is considered as generic as possible. Based on the previous sections on intelligibility and control, our key requirements for such a framework are:

1. User centric: the framework must allow users to define their own context model

2. Abstract: the framework should not restrict itself to specific subtypes of context nor should it be restricted to types of sensor or to any technological constraint at all. It should be possible to derive high-level context entities by combining existing knowledge.

3. Easy to use: since the framework targets end users rather than application designers, it should provide a user friendly interface, with proper attention for usability

4. Expressive: to allow users to define whatever context they can imagine, the framework needs to provide a maximum of expressive power. It has to provide the users with the necessary constructors to build complex context models. The framework needs a good balance between ease of use and expressibility.

## 2.4 Summary

Context is investigated in multiple research areas, varying from human sciences to engineering. The concept has been defined by several researchers but a general accepted definition is not available. The most general definition for context is that it constitutes the necessary conditions for sufficient understanding of phenomena. Context-aware applications are applications that use implicit situational input. These applications use context models to identify the context information they operate on. An important segmentation of context is the separation between physical and logical context. Physical context can be measured from sensors, logical context needs to be derived using complex context models. Human related aspects of context typically are logical context. Context-aware applications that operate on human aspects of context cannot deterministically use pre-defined context models. The user needs to be involved in the processing of context. These context-aware applications need to provide intelligibility and control to users to improve trust and adoption. In order to allow users to create their own context model, a user-centric context modelling framework is required.

# 3

# Frameworks for Context-aware Computing

This chapter discusses context-aware computing from a technical perspective. First, we present some architectural design principles that are generally accepted in the community. Second, we introduce several frameworks that have been created to facilitate the development of context-aware applications. We discuss their architecture and evaluate how they can contribute to the user-centric framework that we want to present as part of this thesis.

## 3.1 Architecture of Context-aware Applications

A lot of research has been done regarding the architecture of context-aware applications. Baldauf et al. [4] have presented a survey where they elaborate on the common architectural principles for context-aware applications. They point out that one of the main principles is the differentiation between the *acquisition* and the *processing* of context information. Only in small-scale context-aware applications with limited functionality, acquisition and processing can be integrated in a single module. For less trivial applications however, these functions need to be separated. Separating acquisition from processing increases flexibility and re-use, and facilitates the use of distributed systems. Chen [21] has provided a similar survey focusing

on context-aware mobile computing research. Further elaborating on this separation of acquisition and processing, Chen [23] presents three different approaches for context-aware applications to acquire context information :

- direct sensor access: Only for small scale applications with limited functionality that have no distributed architecture. Sensors and drivers to operate these sensors are locally built in the components that process this data. An example of such an application is a basic thermostat that senses temperature and activates or deactivates a central heating system by comparing the actual temperature with a desired temperature.

- middleware infrastructure: Context-aware applications can apply a layered architecture to separate low-level sensing details from application logic. Components can be distributed but are not shared between applications.

- context server: Context-aware applications can use a shared context server to handle the sensing and pre-processing of context. The server operates independently of its consumer applications.

In a distributed architecture multiple approaches can be followed to connect components and share data between processes. Winograd [67] presents the following context information processing models:

- Widgets: A software component that provides a public interface to a sensor. Widgets encapsulate the technical details of operating the sensor.

- Networked service: Similar to the context server approach, uses a shared context server where network connected services can be dynamically added or removed.

- Blackboard model: An data centric view where processes can post messages to a central blackboard. This blackboard provides publish-subscribe functions to client applications.

Next to separating *acquisition* and *processing* of context, Dey and Abowd [30] propose to also separate the *interpretation* of context, and the *reasoning* on context. They present a conceptual architecture consisting of five layers:

1. sensor layer: Consists of the physical sensors and data sources that deliver context data.

2. data retrieval layer: This layer uses the sensor layer to retrieve context data. It connects to drivers and APIs of components in the sensor layer.

3. pre-processing layer: In this layer low level context-data is transformed to a higher abstraction level. This transformation process is also referred to as *aggregation* or *composition*. The pre-processing layer is responsible for reasoning on context-data (e.g. by applying rules) and for combining multiple data sources (referred to as *fusion*).

4. storage and management layer: A layer that implements the long term storage of context-data (e.g. using a database) and the retrieval of previously stored data. It provides a public interface to client applications.

5. Application layer: A layer for client applications that consume pre-processed context information. Application responses to detected contexts are implemented in this layer.

As discussed in section 2.1.2, the context model specifies the context information that a context-aware application can use. To maximise re-usability and separation of concerns, context models should have a formal representation that is separated from the rest of the application. Bettini et al. [13] point out that the general functions of context-aware application should not be mixed with the definition and evaluation of context information. Earlier, Strang and Linnhof-Popien [53] already presented a survey on context modelling where they identify generic formats for context models that can be shared between applications. Each format comes with its own tools and methods for validation and processing of context data. Lim and Dey [46] have performed a review to investigate the most popular context-aware reasoning techniques. This review shows that the four most used techniques are rules, decision trees, naïve Bayes and hidden Markov models.

A full review of all context modelling formats an reasoning techniques is outside the scope of this thesis. The framework created as part of this thesis applies rule based reasoning since this provides a good balance between expressive power and runtime efficiency. Rule engines also support dynamic behaviour since rules can be added or modified at runtime. Dynamic behaviour is interesting for user-centric systems where users should be able to modify context-models in a flexible way. In our opinion, the AI-based models provide less control. This is also illustrated by Youngblood [68] who presents a context-aware system based on Hidden Markov Models where he admits that the user occasionally needs to bypass the system because it lacks sufficient flexibility and control.

To facilitate the development of context-aware applications, multiple frameworks and tool kits have been presented. Examples are JCAF [6], Solar [22] and Context Toolkit [30]. Baldauf et al [4] present several frameworks in

their survey. As pointed out by Dey and Newberger [31], these frameworks address some of the common challenges of context-aware applications:

- using a distributed set of resources
- asynchronous message communication and event subscription
- resource discovery
- platform independent communication protocols

Taking away the need to explicitly program these technical functions, developers can focus on end user functionality. The component based architecture of these frameworks promotes reuse of components (e.g. components encapsulating sensors) and facilitates extensibility.

## 3.2 Context Toolkit

### 3.2.1 Architecture and Components

One of the best known frameworks for building context-aware applications is the Context Toolkit presented by Dey and Abowd [30] in 2001. This conceptual framework separates the acquisition and representation of context from the delivery to and the response from applications. It implements most of the design principles mentioned in section 3.1. According to Dey and Abowd, the architecture for a context-aware framework should implement:

- Separation of concerns: Frameworks should promote the use of independent reusable components.
- Context interpretation: Frameworks should support the transformation of low-level context data into high level context information that has a meaning for applications.
- Transparent, distributed communications: Since context-aware applications typically have a distributed architecture, communication between components needs to be implemented. Frameworks should take care of the implementation of communication protocols.
- Constant availability of context acquisition: Context-acquisition modules are independent modules, shared by several client applications. They cannot be instantiated or controlled by the applications using them.

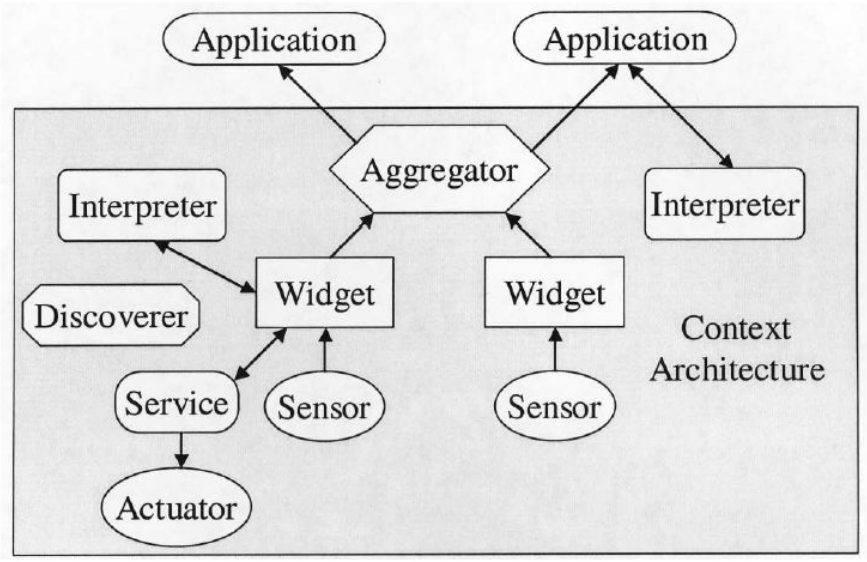


Figure 3.1: Context Toolkit Components [Dey, Abowd, 2001]

- Context storage and history: This should not be a responsibility of the application, but of the components that (pre-)process the context information, so that historical context data automatically becomes available for all client applications.

- Resource discovery: In a distributed and dynamic environment, modules can be added and removed. The framework should provide support for the automatic adaptation to changing resources.

The authors have instantiated this conceptual framework in a toolkit: the *Context Toolkit*. This toolkit provides built in features that support the rapid prototyping of context-aware applications. In order to cover the above architectural requirements, Dey and Abowd have created abstractions for the separate functions involved in the acquisition and processing of context: widgets, interpreters, aggregators, services, and discoverers. Figure 6.3 shows how these components cooperate.

**Context widgets** are the central components of the architecture. The name *widget* comes from an analogy with widgets in graphical user interfaces. In traditional GUI systems, user interface widgets are reusable components that mediate between users and applications. These components implement low-level details regarding the graphical presentation and operation of the widget, that are hidden for the application using the widget. Application developers use the widgets interface to provide input or process output that

is delivered through the widget. Dey and Abowd have copied this approach to context acquisition by introducing context widgets. A context widget is a module that connects to sensors and captures information from these sensors. Technical details on how to operate the sensors are encapsulated in the widget. Widgets have a state and a set of methods. They make the acquired context information available to clients through a public interface that allows both pull and push mechanisms. A consumer of context information can pull information using synchronous API calls. Push mechanisms are implemented using a classical publish-subscribe pattern. Consumers can subscribe to a selection of events in order to be notified by the widget when such events are detected.

**Interpreters** provide a mapping function, they transform information. In this transformation process, interpreters can apply complex logic to raise the level of abstraction of context information. Interpreters are stateless, they simply provide processing functionality. Interpreters expose a public interface that can be used by any other component: widgets, aggregators, applications, etc.

**Aggregators** collect logically related context information from multiple widgets. Given the distributed nature of context-aware systems, applications typically integrate context information that is acquired by multiple sensors. An aggregator would connect to all relevant widgets, and combine their information, possibly raising the level of abstraction of this data into more useful information. This information is then made available to clients through a public interface, similar to the interface that widgets provide.

**Services** provide a interface to *act* upon the environment. Whereas context widgets can be seen as input devices, services are output devices. They hide low level implementation details on how to perform an action, and provide a public interface.

**Discoverers** are responsible for managing the distributed environment. In such an environment, components can be dynamically added or removed. The discoverer keeps track of all active components and maintains a list of the functions they provide. When an application needs context information, it queries the discoverer that responds with a list of components that are currently active and provide the required functionality.

**BaseObject** is an abstract class that implements communication functionality. In the Context Toolkit all components need to be implemented as subclasses from this abstract class.

### 3.2.2 A Sample Widget

Dey and Abowd have demonstrated the use of their *Context Toolkit* in various applications, most of them involving location as part of the context. To illustrate the use of the Context Toolkit, we discuss the *IdentityPresence* widget presented by Salber, Dey and Abowd [49]. This widget controls a sensor deployed at a specific location that senses the environment for the presence of people. The sensor can also identify the people it detects. Identifying nearby people can be useful for surveillance applications, tour guides, call forwarding tools etc. As mentioned in previous section, a widget has a state and a behaviour. The state can be queried at the client's initiative in a synchronous request, the behaviour consists of possible callback functions that the widget can invoke when specific events are detected. The state consists of three attributes: the location that the widget is monitoring, an identifier for the last person it sensed, and the most recent time it detected a person arriving. The behaviour consists of callback functions related to two types of events: a person entering the area controlled by the widget and a person leaving this area.

Multiple types of sensor can be used to implement the detection of these events (examples are cameras with image recognition software, badge detection systems, transponder devices, etc.). These different implementations are transparent for the applications using the widget. They can be replaced at runtime and applications can even aggregate information from widgets with different implementations. The authors have used the *IdentityPresence* widget for building several straightforward context-aware applications:

- In/Out board : A board that indicates whether people are in a building or not.
- Information Display application : Personalized information is shown to a user on a nearby display.
- DUMMBO meeting board : An application that captures information from meetings that occur spontaneously at a specific location.

### 3.2.3 Evaluation

From an application designer's perspective, the *Context Toolkit* is a powerful tool to implement context-aware applications. It has a modular design and promotes reusable components. The toolkit enables rapid prototyping and has been successfully applied in various projects. Salber et al. [49] added context awareness to existing applications using the toolkit. They found that this required only minimal code changes to these existing applications.

However, the Context Toolkit is only the start in building context-aware applications. The framework offers little support for composing context information items or reasoning about context. As no templates or other composition mechanisms are available, all composition needs to be implemented by coding. It is useful for straight-forward applications that do not require complex reasoning on context but not for applications that work with complex human related context. It misses explicit formalised context models, which was already acknowledged by Dey and Abowd [30] who admit that more attention needs to be given to knowledge modelling and representation. A similar remark was made by Bradley [15] who indicates that the framework was implemented for very primitive applications only. He points out that the Context Toolkit misses functionality to support codification of complex contextual details since the context model is mixed with other functions. No separate algorithms to manage inference are provided, even the interpreter component is only an abstraction that needs to be implemented by coding. The lack of a separate context model also makes it difficult to implement dynamic behaviour.

In our opinion, it should be possible to create an additional layer that implements a formal context model. This layer would collect callbacks from several widgets and compose these according to the context model. This would allow designers to reuse some of the components of the framework, while adding dynamic behaviour and support for more complex context models. However, as most context-processing would be implemented outside of the original Context Toolkit, this approach would reduce it to a simple set of low level delivery channels.

Bellotti and Edwards [9] provide a more fundamental remark. They argue that a component based setup cannot provide intelligibility. In a component based setup the encapsulated modules that acquire context data cannot directly communicate with application users. As a consequence there is no possibility to provide information to users on how the system acquired context information and how it processed it. This results in applications with poor intelligibility. Bellotti and Edwards also claim that all reasoning on context must be situated in the application itself, because that is where the relevant semantics reside. Dey has modified the original Context Toolkit to provide support for intelligibility [29]. In this version, widgets get additional functionality to communicate directly to users. In our opinion however, this conflicts with the original design of Context Toolkit that focuses on encapsulation and reuse. We therefore agree with Bradley's comment [15] that the framework is best fit for straightforward applications with a simple static context model.

## 3.3 Adding Support for Intelligibility and Control

In section 2.2 we elaborated on the importance of intelligibility and control for context-aware applications. Supporting intelligibility and control has a significant impact on the adoption of context-aware applications [7]. As discussed in above section, Bellotti and Edwards [9] pointed out that separating context acquisition, reasoning and response results in poor intelligibility. Existing context-aware frameworks facilitate the creation of context-aware applications but they do not provide support for intelligibility and control.

### 3.3.1 The Situation Component

Dey and Newberger [31] propose to extend existing frameworks for building context-aware applications to explicitly support intelligibility and control. They argue that intelligibility and control must be addressed at user interface level since they are important for end users. This implies that a system exposes its application logic (e.g. the context-aware rules) and underlying infrastructure (e.g. the kind of sensors that are used) to end users. Intelligibility and control means *accessing* and *manipulating* application state. Since context-aware applications typically have a distributed state, information from various components needs to be integrated before presenting it. This requires components that support intelligibility and control to be central in the architecture of a context-aware system. Dey and Newberger [31] introduce such a component called *Situation*. Situations are components that encapsulate application logic and provide an API to inspect and manipulate application state. They organise applications in a similar way as component architectures such as JavaBeans. The application logic of a context-aware application typically consists of a set of context rules (e.g. when a user enters a specific location, perform an action). A situation encapsulates such a context rule, takes care of the acquisition of context, processes this information and triggers the application response. Developers only have to provide the context rule using a declarative mechanism. The details of resource discovery, subscriptions, and the issues of dynamically appearing and disappearing components at runtime are handled by the situation component. This is accomplished by internally decomposing context rules into *references*, *parameters*, and *listeners*:

- **References:** Situations encapsulate a rule. To validate whether the condition of this rule is satisfied, specific context information needs to be verified. To acquire this context information, situations create

*references* to information sources. When Context Toolkit is used as underlying infrastructure, these information sources are abstracted as *widgets*. References use a discover to check what information sources are available, and subscribe to the relevant events.

- **Parameters:** Situations automatically extract parameters from a rule when it is added. These parameters are similar to JavaBean properties, and can be used to expose and manipulate application state.

- **Listeners:** Listeners are notified when state changes occur in the situation component. This can happen as a result of context data being received from the information sources, or when a user manipulates state using parameters. Listeners are also notified when actions are executed as a consequence of a rule being triggered. Listeners provide a central point for collecting information to support intelligibility.

Dey and Berger [31] have implemented situations on top of Context Toolkit since it is the most commonly used context-aware framework. The authors claim however that any framework can be used as long as it supports resource discovery, context input components an service actuators. The implementation also includes modules in Adobe Flash and Visual Basic that interface with situation listeners and can be used to build user interfaces that present application state and provide functions to modify application state.

### 3.3.2 Evaluation

While situations only support rule based application models, the toolkit was later extended by Lim and Dey [46] to include support for Decision Trees, naïve Bayes classifiers and hidden Markov Models. Some types of explanations however remain unsupported: intelligibility on history is not available. All intelligibility types apply to current state, while application behaviour can also be influenced by long term history. Control is supported but in a limited way. Although parameters can be updated to modify state, the application behaviour itself cannot easily be modified. Rules cannot be added or modified at runtime.

## 3.4 Context Recognition Network Toolbox

The Context Recognition Network Toolbox presented by Bannach et al. [5] is a framework designed for rapid prototyping and fast implementation of context-aware applications. It provides a graphic interface to assist designers.

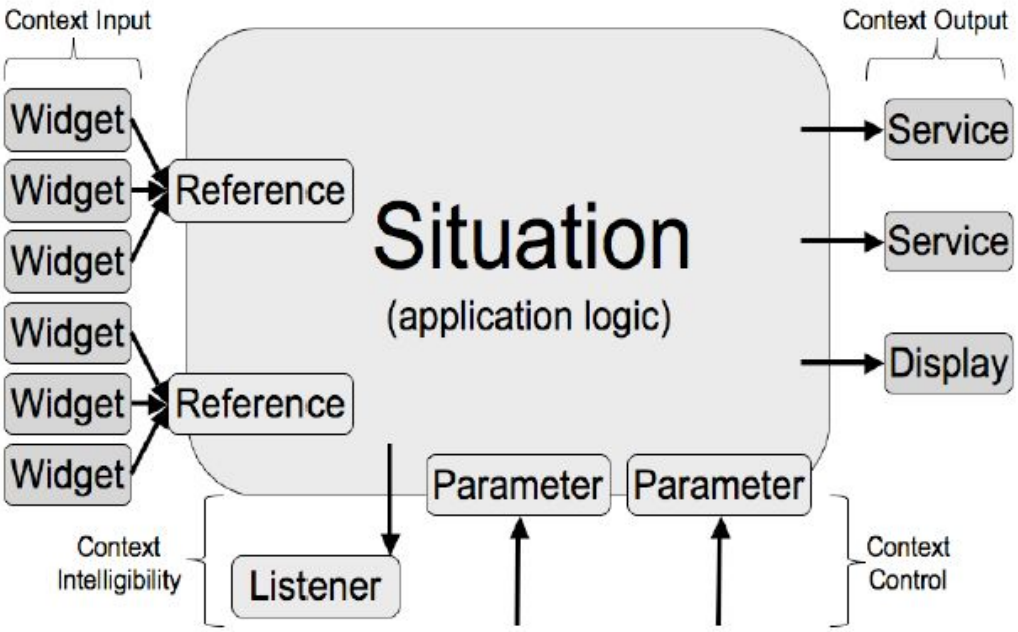


Figure 3.2: A situation component implemented on top of Context Toolkit [31]

Multiple predefined modules are available for the processing of context data. The Toolbox is designed for the recognition of end user activities. Attention is given to an efficient runtime behaviour, and support for several target platforms is provided.

### 3.4.1 Architecture and Components

The Context Recognition Network Toolbox helps designers to construct processes that recognize activities based on sensor input. It provides a repository of parametrizable software components that are ready to use without additional coding (illustrated by (1) in Figure 3.3). A Gui (illustrated by (2) in Figure 3.3) is available for designers to compose activity recognition processes. The Toolbox also provides a runtime environment for various platforms (illustrated by (3) in Figure 3.3). Various components exist for interacting with external applications (illustrated by (4) in Figure 3.3).

The Context Recognition Network Toolbox applies a data-flow driven design. A designer can define an activity recognition process by composing *tasks* and *data packets*. A task is an active object that operates on data streams. Tasks operate on data packets they receive at an in-port and gener-

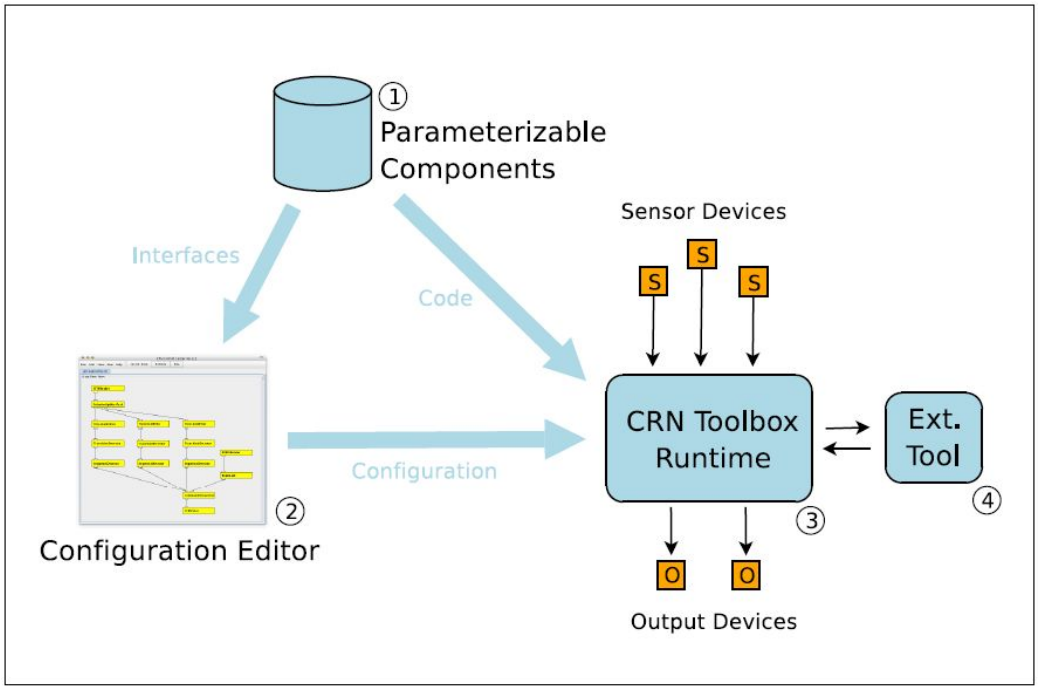


Figure 3.3: CRN Toolbox [5]

ate new data packets at an out-port. Tasks are connected by data flows that start at the out-port of a first task and end at the in-port of a second task. Multiple types of tasks are available to construct the needed functionality:

- reader: A task from which a data-flow originates. It has no in-port. Readers typically encapsulate sensors. Multiple generic (e.g. reading from file or TCP socket) and specific (e.g. reading from a Wiimote or various RFID devices) readers are available.

- writer: A task at which a data-flow ends. It has no out-port. Writers provide a gateway to external systems. Available writers are a TCP writer or a file writer.

- filter: A task that filters information from data-streams. Available filter plug-ins include average signal energy, bandwidth, max, mean etc.

- classifier: A task that classifies input data. The toolbox offers support for several classification algorithms (e.g. Hidden Markov Models, K-Nearest Neighbour, Probabilistic Context-Free Grammar parsers, etc.).

- merger: Merge multiple data-streams according to a selected algorithm.

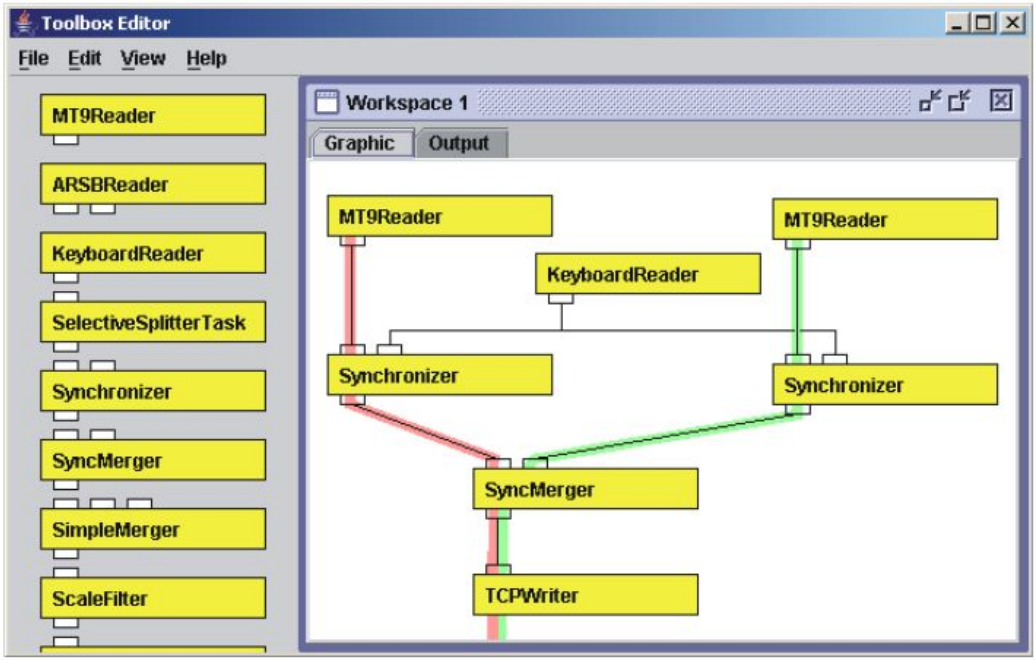


Figure 3.4: CRN Toolbox Editor [5]

- splitter: Split data-streams according to a selected algorithm.
- synchroniser: Synchronise data-streams based on a synchronisation event.

An example of the designer GUI is shown in Figure 3.4. Using this GUI, a designer drags available components in a working area. Settings for parameters of tasks are specified, and tasks are connected by streams. These activities do not require coding. If all needed tasks are available, an activity recognition application can be constructed without writing code. Banach et al. illustrate this by presenting an application that recognizes kitchen activities based on a motion sensor. No coding was required.

### 3.4.2 Evaluation

Bannach et al. [5] illustrate the successful use of the Context Recognition Network Toolbox by providing a list of projects that applied this Toolbox. They also mention a case study where students where able to complete an activity recognition process within 20 hours. We agree that the Toolkit is well suited for recognizing simple activities based in physical context. It offers a complete solution thanks to the direct support for fusion of multiple

data sources, data manipulation and classification algorithms. The GUI for composing processes is a step in the direction we need, but it remains a tool for designers that is too complicated to be given to end users. The absence of separate context models prevents complex reasoning on context. Our conclusion is that the framework was not designed to be user-centric, and does not support complex context models.

## 3.5 The Jigsaw Project

### 3.5.1 The Framework

Most frameworks for context-aware applications do not support end user programming. One of the exceptions is the Jigsaw project presented by Humble et al. [39], a framework for context-aware computing in domestic environments. The framework includes a GUI for end users that can be used to compose components (called *transformers*) into a process. These components represent various devices that are present in a smart home (e.g. motion sensors, cameras, switches). By composing these components in a process, a user can program their behaviour.

The implementation of this component model is based on a shared data space. A real world device can be added to this shared space by creating a JavaBean that represents it. This JavaBean typically includes driver information, implements the required interface and defines the properties that are made available to the framework. Components are called *transformers*. Three main classes of transformers are available:

- Physical to Digital transformers measure a physical attribute using a sensor. The measured value is transformed into a digital property that is then shared through the dataspace.
- Digital to Physical transformers can trigger an action on a physical output device based on values of shared digital properties.
- Digital transformers process digital information. They increase the level of abstraction.

When transformers are registered to a shared data-space they become available in the configuration GUI. This GUI is based on the metaphor of assembling jigsaw pieces and is therefore called the *Jigsaw Editor* (see Figure 3.5). Users can create a process by connection jigsaw pieces sequentially from left to right. Connecting pieces is accomplished by drag-and-drop manipulations on a canvas. When a piece is selected for dragging, other pieces that have

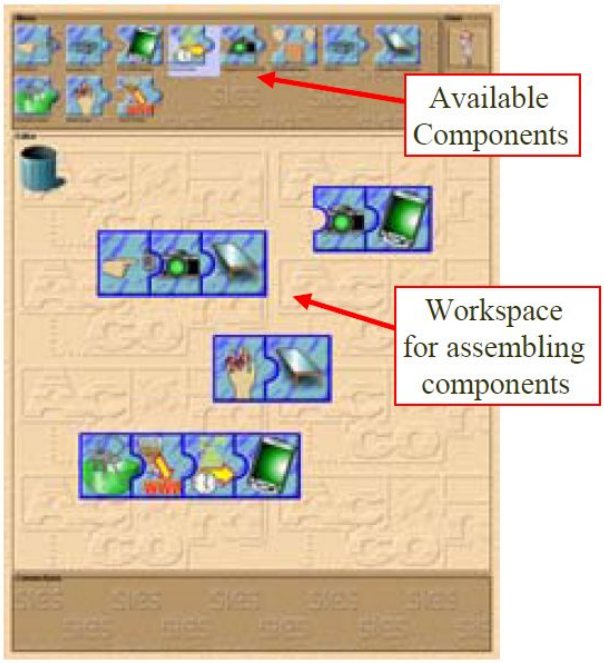


Figure 3.5: The Jigsaw Editor [39]

compatible attributes are enabled. The piece can then be connected to any of these enabled pieces. If multiple combinations of attributes can be connected, a dialogue window opens in which the user can select the desired property.

### 3.5.2 Evaluation

The Jigsaw project provides a user friendly configuration GUI for end users. This GUI however is not very expressive. Humble et al. [39] already acknowledge that their framework is more suited for understanding and changing the configuration of smart devices in a home then to fully program such a configuration. The framework is focused on straightforward implementations of context. Since no separate context models are used it is difficult to implement complicated context reasoning. While it is useful in a restricted environment such as home automation, it is difficult to apply to a more generic environment.

## 3.6 Summary

In this section we have discussed the architectures of context-aware applications. Most researchers prefer layered architectures that separate the acquisition, interpretation, and reasoning on context. However, care needs to be taken to provide sufficient intelligibility and control when applying a component based architecture. Many frameworks and toolkits have been presented to facilitate the development of context-aware applications. We have presented the Context Toolkit, the Context Recognition Network Toolkit and the Jigsaw project. These frameworks provide fast prototyping and come with built in context acquisition and processing tools, but they either lack control and intelligibility, or are too specific for our purpose.

# 4

# The Context Modeller Framework

This chapter introduces the Context Modeller framework, our main contribution in this thesis. We give a high level overview of the framework and its layered structure. We then focus on the provided tools and explain how they are combined to model context.

## 4.1 Custom Types

### 4.1.1 Creating Types

Creating context models implies working with context information items. These information items have specific data types. Our framework does not predefine these data types. To be as generic and extensible as possible, all data types used in context models have to be defined using the framework. A data types is either a primitive type, or a composed type. Primitive types are data types known by the runtime environment (e.g. Float, String) that do need a definition. Composed types are data types created by a user for which the runtime environment needs an explicit definition. As explained in the following sections, composed types can be created and used in multiple layers of the framework. A composed type has a name and an ordered set of attributes. An attribute has a name and a data type which can again be a primitive type or a composed type. The only restriction when creating

composed types is that a name for an attribute can only occur once in the same composed type. Figure 4.3 shows the composed types `Located` and `Location` as they are presented by one of the tools of the framework. Both composed types have two attributes. To be as generic as possible, primitive data types are not hard coded in the framework and have to be specified.

### 4.1.2 Type Compatibility

When combining information items in the modelling process, the data types of these items need to have a degree of compatibility. As we will explain in the following sections, to create a context model a user needs to select context information items and use them as input for processing steps. Whether a context information item can be applied or not as input depends on its data type. The data type of the context information item has to be *compatible* with the input data type for this processing step. This is our definition of type compatibility:

> *A data type is compatible with a second data type if its set of attributes is a superset of the set of attributes of the second data type.*

Remark that this relation is not symmetric. Since attributes have a name and a data type, this definition implies that in order for a first data type to be compatible with a second data type, for every attribute of this second data type the first data type needs to have an attribute with the same name and data type. Figure 4.1 illustrates type compatibility. On the left it shows two data types *Located* and *MeetingRoom* with their attributes. On the right it shows two data types that can be used as input type for a processing step. The arrows show the type compatibility relation. Remark that the `Located` data type is not compatible with the second input data type, since it does not have an attribute with name `Name` and type `String`.

By defining type compatibility in terms of attributes rather than simply requiring the types to be equal, the framework is more flexible. Information items with multiple types can be applied as input to the same processing step. When defining the processing steps, users do not have to be aware of the actual data types for the input events, they only have to specify the attributes relevant for the processing step. In the following sections, we will refer to a processing step as a *template* and an input data type as the type of an *input connector*.

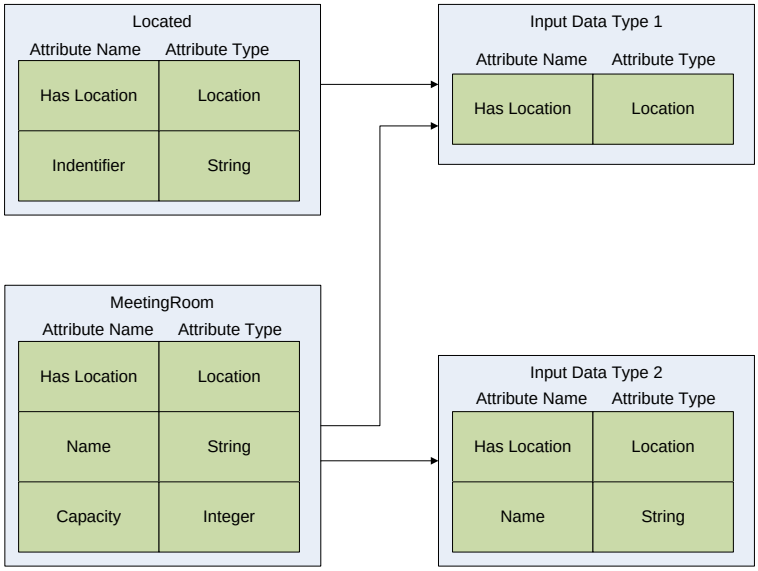


Figure 4.1: Type compatibility

## 4.2 Layers of Abstraction

In previous chapters we argued that end users need to be involved in the creation of context models. Since end users can not be expected to have programming skills, the creation of these context models needs to be handled by a tool that offers users a straightforward and understandable interface. At the same time, we do not want to restrict the expressibility of the framework. To cope with the dilemma of usability versus expressive power, we have created a framework that consists of three layers of abstraction. Figure 4.2 shows these layers. Every layer covers a part of the modelling activities, produces different deliverables and requires users with different profiles. In the following sections, we describe these layers in detail.

### 4.2.1 The Plug-in Layer

In order to create a context model for a context-aware application, designers need to be aware of context information the application has access to. In our framework, the acquisition of context information is handled in the Plug-in layer. A plug-in is a module that detects a specific type of context-information. It connects to data sources (e.g. sensors, databases, external services) and retrieves information that it transfers to the runtime environment of the framework as plain data objects with a custom defined type. Information items delivered by a plug-in at runtime are referred to as *dy-*

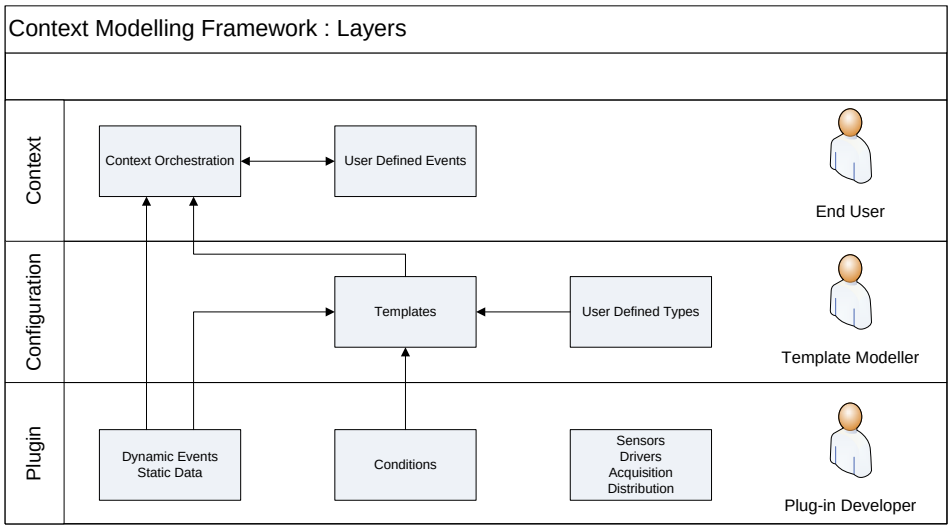


Figure 4.2: Layers of abstraction

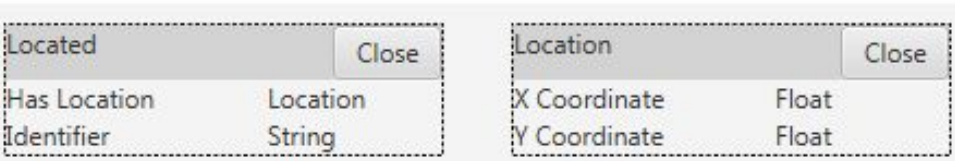


Figure 4.3: Composed data types used by the `Locator` plug-in as visualised in the framework

*namic events*. We illustrate the concept of a plug-in with the example of the *Locator* plug-in. This plug-in can detect persons at a specific location. When the plug-in detects a person at a specific location, it creates a *dynamic event* with data type `Located`. The data type `Located` has an attribute with data type `String` to identify the detected person and an attribute with data type `Location` to indicate the location at which the person was detected. The data type `Location` has two attributes of type `Float` to represent its Cartesian coordinates. Figure 4.3 shows the data types `Located` and `Location` as they are presented by one the tools of the framework. In our example we consider the data type `Float` to be a primitive type of the runtime environment.

Next to the implementation of the data detection modules, the plug-in developer has to provide definitions for all *dynamic events* including their data types. Other data that can be included in the specification of a plug-in is a set of pre-defined objects that can be used in context models. For the example of the `Locator` plug-in, a useful set of pre-defined objects would be

a list of places of interest with their location (e.g. the study room, the living room, various meeting rooms at the office). Pre-defined objects are referred to as *static* data: they are loaded in the runtime environment at start up and cannot be modified.

Another responsibility of the plug-in developer is the definition and implementation of *conditions*. A *condition* is a boolean function that takes a list of attributes as input. Conditions provide a mechanism to implement reasoning on the attributes of context information items. In the example of the *Locator* plug-in a condition *inRange* is provided, that takes two attributes of type `Location` as input. It calculates the Cartesian distance and returns *true* is this distance is below a specific threshold value.

To specify *dynamic events*, *static data* and *conditions*, a plug-in developer needs to create files in JSON format. To implement conditions, a developer needs to provide the code required for their runtime evaluation. As these task are handled by developers with programming skills, no supporting tools are provided.

### 4.2.2 The Configuration Layer

The second layer provides a mechanism to raise the level of abstraction of context information called *templates*. These templates combine the context information items provided by the plug-in layer to create higher level context information items. Context information items that are combined in a template are referred to as the *input events* of the template. A template can apply conditions to these input events and generate an *output event*. When the runtime environment of the framework detects that all input events are present such that the attributes of these input events make the conditions evaluate to *true*, it generates the appropriate higher level output event. We explain the three parts of a template more in detail:

- **Input events:** There are two categories of input events. It can either be an information item provided by a plug-in such as a dynamic event or a static data object, or a more generic place holder. A place holder is simply a custom data type. If the input event of the template is a place holder, every context information item with a data type that is compatible (as defined in section 4.1.2) with the data type of the place holder can be applied to it.

- **Conditions:** Conditions are provided by plug-ins. A condition can be added to a template, by linking its attributes to the attributes of the input events. These linked attributes will be used to evaluate the condition of the template at runtime.

- **Output event:** A template has a single output event. This output event can be specified in two ways. It can either be defined in the template by specifying its list of attributes. In this case every attribute of the output event needs to be explicitly linked to an attribute of one of the input events. Another possibility to specify the output event of a template is not including its structure in the definition of the template. In that case the structure of the output event is derived automatically from the applied input events. This is useful in case place holders are used for input events, since the structure of the input event is not defined yet in the template. When the output event is derived automatically, it will copy all attributes from all input events to the output events with the restriction that an attribute name can only be used once. If the same attribute name is used in multiple input events, only the first encountered attribute will be copied to the output event. The first possibility to specify output events offers more control to the template developer, the second possibility provides more flexibility, since the structure of the output event is not fixed in the definition of the template.

As part of our POC we implemented the template illustrated in Figure 4.4. This template has two input events. Both input events are place holders with only a single attribute for a location. This implies that all information items that have this attribute can be applied to this template. The template applies the *inRange* condition to the location of its input events. The output event was not explicitly defined, implying that this template can generate output events with different types, depending on the actual input information items. The template is applied to the dynamic event `MyLocation` and the static data object `MyStudy`, as illustrated in Figure 4.5. The template generates an event with name `inStudyRoom`, that inherits the `Has Location` attribute from the first input event, and the `Has Name` attribute from the second input event. Note that the `Has Location` attribute from the second input event is not transferred to the output event since an attribute with that name already exists in the output event.

The definition of *templates* is handled by expert users that have a good understanding of the framework. They need to be aware of the dynamic events a plug-in provides and the conditions that can be applied to their attributes. Expert users need to have an idea of the context data items an end user is interested in and how end users want to combine these items. The framework provides expert users with a tool for the creation of templates: the *ExpertUser Tool*. Since this tool has a graphical interface, expert users do not need programming skills.

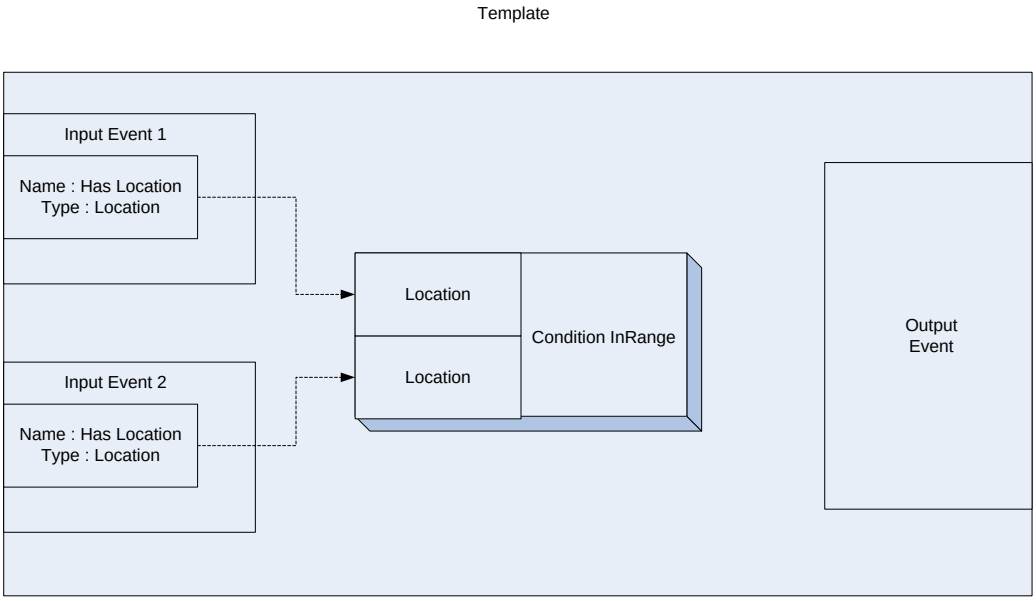


Figure 4.4: Template `Nearby`

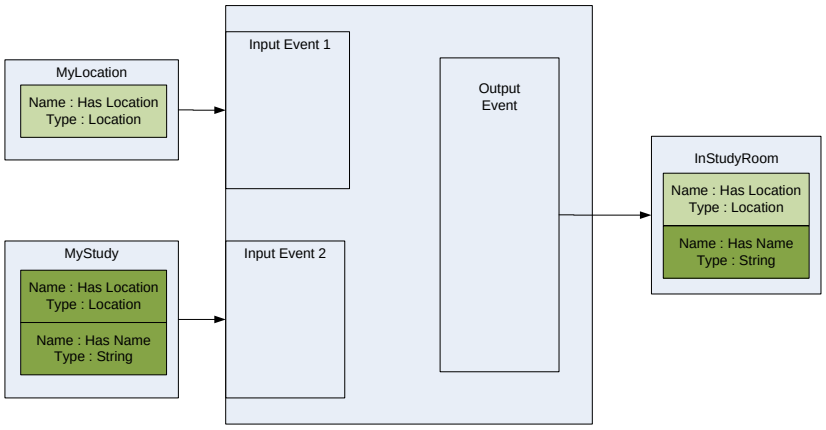


Figure 4.5: Template `Nearby` applied to `myLocation` and `MyStudy`

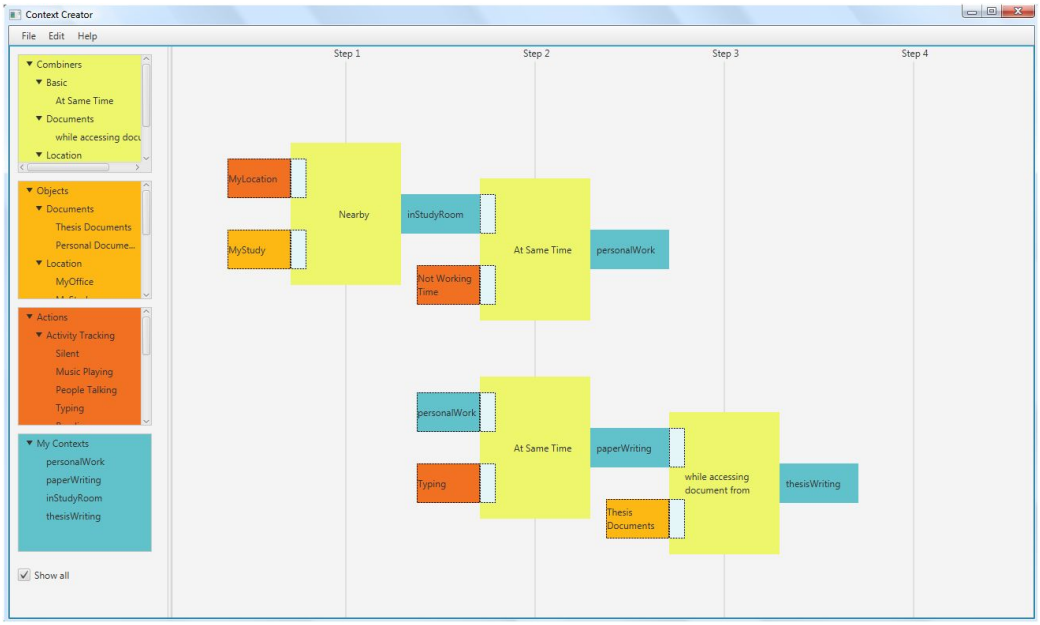


Figure 4.6: EndUser Tool with orchestration to derive the context `Working on Thesis`

### 4.2.3 The Context Layer

The creation of a context model is managed in the context layer. This is the layer for end users. They combine the templates created in the configuration layer to create orchestrations. An orchestration specifies how the combination of several context information items derives a high level context that has meaning for the end user. Since templates are used, the expressibility of these orchestrations is restricted to what the templates provide. Therefore, it is important that the template developers have an understanding of end users needs, and that templates are defined as generic as possible. Figure 4.6 shows the EndUser Tool with an orchestration created in the workbench. We will explain the EndUser Tool more in detail in section 4.4.2.

## 4.3 Context Evaluation

The actual evaluation of context at runtime is implemented with a rule engine. A rule engine has a knowledge base that consists of a set of rules and facts. The framework translates a created orchestration to a specification in a declarative language that is loaded into the knowledge base. In this declarative specification every template in an orchestration is translated into

a declarative rule. Below code snippet shows the result of this generation on the `Nearby` template illustrated in 4.5:

```
package office

rule "rl_inStudyRoom"
when
$1 : MyLocation()
$2 : MyStudy(inRange($1.HasLocation , $2.HasLocation ))
then
inStudyRoom output = new inStudyRoom();
output.setHasLocation($1.getHasLocation());
output.setHasName($2.getHasName());
insert(output);
end
```

Static data objects are loaded as facts in the knowledge base at start up of the runtime environment. The runtime components of connected plug-ins deliver dynamic events to the rule engine. These dynamic events are also added as facts to the knowledge base. Every time the knowledge base is modified, the rule engine evaluates its rules and executes the rules that are satisfied. For the rules generated by our framework, execution of a rule means that the user defined output event is generated and added as a fact to the rule engine.

## 4.4 Working with the Tools

In this section, we explain the use of the tools created as part of the framework to create templates and orchestrations. We have provided the ExpertUser Tool for the creation of templates, and the EndUser Tool for the creation of orchestrations.

### 4.4.1 The ExpertUser Tool

Based on the dynamic events, conditions and static data objects provided by the plug-in specifications, an expert user can create templates using the ExpertUser Tool. This Tool consists of a central workbench area and four lists: *Templates* and *Compositions* at the left side of the workbench, and *Conditions* and *Built in Types* at the right side of the workbench. When the tool is initially launched only the two lists at the right side are populated. These two lists are populated with information from plug-in specifications. Items from these lists can be used but not modified by the expert user. The *Conditions* list shows an entry for every condition defined in the plug-in. The *Built in Types* list contains three types of data:

1. composed data types of the dynamic events that a plug-in can deliver

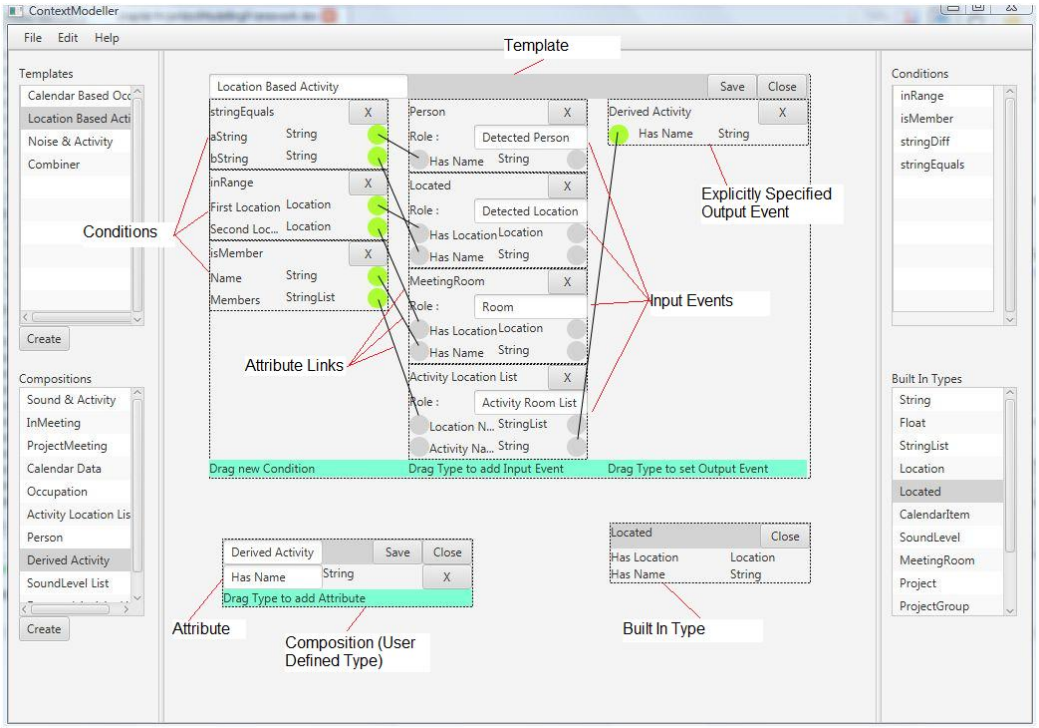


Figure 4.7: ExpertUser Tool with a template, a user defined type and a built-in type in the workbench

2. primitive data types supported by the runtime environment of the framework (e.g. Float, String)

3. static data objects delivered as part of a plug-in specification

The two lists at the left side are populated by the expert user using the ExpertUser Tool. The *Compositions* list contains user defined data types that can be used to create place holders as input events, or to explicitly define the structure of the output event of a template. These user defined data types have a name and a set of attributes. Every attribute has a name that can be entered as text and a data type that can be selected from the *Compositions* list, or from the *Built in Types* list.

The most important list is the *Templates* list which shows all user created templates. The user can create *Templates* and *Compositions* by clicking the *Create* button below the appropriate list, and dragging the mouse to a position in the workbench. The user can update existing *Templates* and *Compositions* by dragging the entry from the appropriate list to the workbench. Remark that entries from the *Built In Types* list can also be dragged

to the workbench, but they cannot be modified. Figure 4.7 shows the workbench with a template, a composite type `Derived Activity`, and a built in type `Located` (as a read only object). Remark that this template explicitly defines its output event with a single attribute `Has Name`.

As mentioned in Section 4.2.2, a template combines input events with conditions to generate an output event. Every attribute from every condition needs to be linked to an attribute from one of the input events. Linking attributes is done by dragging the connectors (green circles in Figure 4.7) from the conditions to the connectors (grey circles in Figure 4.7) of the attributes of the input events. A link is shown as a straight line from one connector to another. Links can only be created between attributes of the same type. In case of an explicitly set output event (as in the template shown in Figure 4.7) the linking of its attributes is done in a similar way.

When all templates are created, the expert user can save his work in a *package file*. This file can be opened again in The ExpertUser Tool for further modifications, or can be used as an *input package* for the EndUser Tool as explained in the next section.

### 4.4.2 The EndUser Tool

In the EndUser Tool, a user can create orchestrations by graphically combining objects in a central workbench. Figure 4.8 shows the EndUser Tool with a single template in the workbench. At the left side of this workbench, the tool has four boxes to present the objects that can be used in the workbench for the modelling task :

1. Templates: Defined in the ExpertUser Tool, templates provide a mechanism to generate user defined events based on input events.

2. Static Data Objects: Information about the world that the framework knows about (e.g. persons, rooms).

3. Dynamic Events: Information about the world that the plug-ins can detect.

4. User Defined Events: Starting from a template, users can define the events that they consider relevant.

These four boxes have a different colour. Objects in the workbench have the same colour as the box where they originated from. The EndUser Tool relies on packages created in the ExpertUser Tool. The information from these packages is presented in the first three boxes. These boxes present a list of

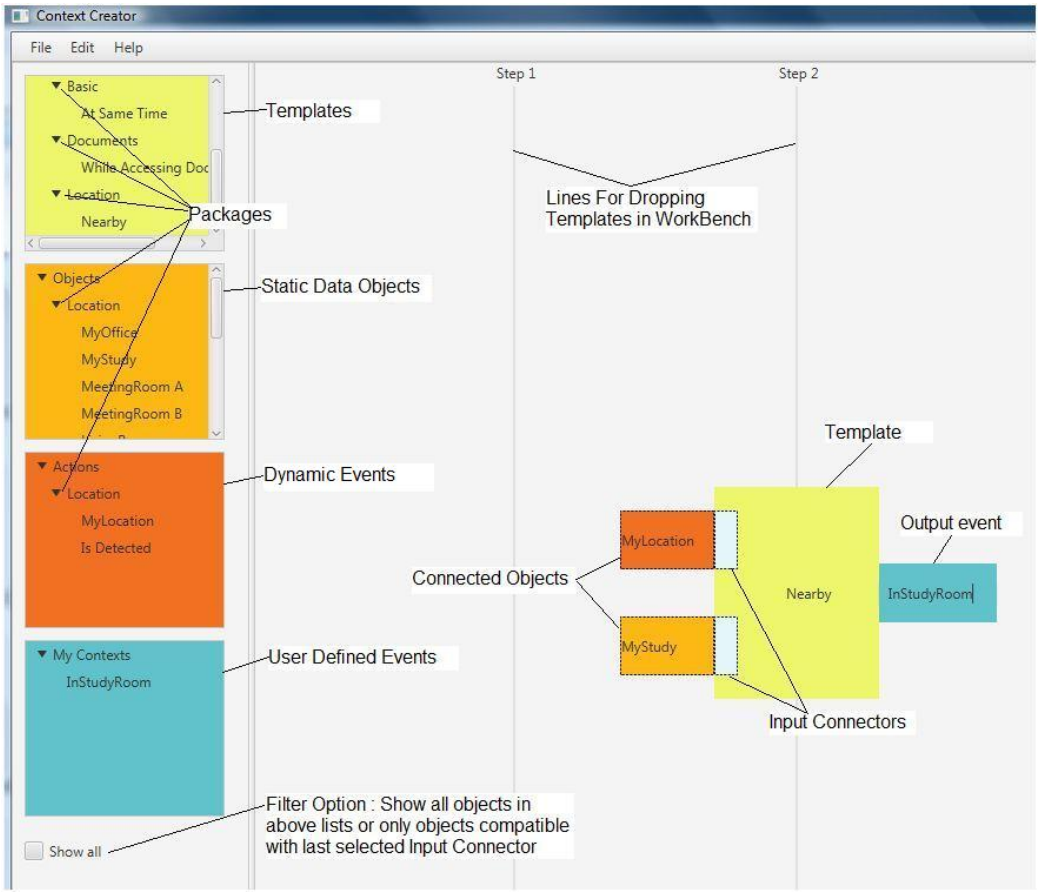


Figure 4.8: EndUser Tool

loaded packages, where every package in the list can be expanded to show the objects in the package. The fourth box *User Defined Events* is populated during the creation of an orchestration: when a user sets the name of the output event in a template, this name is added to the list of *User Defined Events*.

When a user selects the 'new' item in the menu, the application loads all package files it can find at a specified location. When the orchestration is created, it can be saved as a file, or compiled into a set of declarative rules that can be loaded in the rule engine. It is also possible to load previously saved orchestration files again in the EndUser Tool for further modification.

An orchestration consists of sequences of templates. As explained earlier, an input event of a template can be fixed in the definition of the template (if a detected event or static data object is used in the template), or can be a place holder that can later be applied to any compatible context information item. In the EndUser Tool, a template is presented with an input connector for every input event defined as place holder. Setting the actual input event is done by dragging a compatible object to an input connector. Input events that are not set as place holders in the definition of the template are not visible in the EndUser Tool.

A user also has to specify the name of the output event generated by the template. When a name is set for the output event, it occurs in the list of *User Defined Events*, where it can be used as input event for other templates. Remark that the template hides a lot of information for the end user: conditions, input events not based on place holders, and all attributes and bindings. When using templates in the EndUser Tool, a user is only aware of:

- The name of the template. This is not modifiable and used to identify the template.
- Input connectors for input events defined as a place holder. Every input connector consists of a set of attributes. This set is not permanently visible to end users, but can be revealed to the user as a tool-tip when the user hovers over the input connector.

To add a template to an orchestration, the user selects a template from the list and drags it into the workbench of the application. The template can be dropped at one of the vertical lines in the workbench. After dropping the template in the workbench the user can link compatible objects to its input connectors by selecting entries from the *Static Objects* list, the *Dynamic Events* list, or the *User Defined Events* list. Every input connector will check the object for compatibility. When the application is in filter mode (i.e. the

*Show All* check box is not selected), only objects that are compatible with the last selected input connector are shown. As an alternative way to connect a user defined event to an input connector, a user can select a template from the list and drag it immediately to the output connector of an existing template in workbench. This results in the new template being graphically attached to the output connector of the previous template, suggesting a sequential flow.

To improve the quality of the visual representation of orchestrations, the workbench enforces certain constraints. Events in the workbench are always linked to a connector of a generator module, they cannot be dropped just anywhere. Templates can only be dropped at the vertical lines in the workbench, constraining the horizontal position of templates in the workbench. After dropping the template, its horizontal position is fixed. The vertical position of elements in the workbench is not fixed, by selecting an element in the header position, it can be moved up or down. In such case, all elements linked to each other move as a single group. Templates can be removed from the workbench by a right click on them. The template will only be removed if no other template is directly connected to its output connector. When a template is removed, objects connected to its input connectors are also removed.

## 4.5 Summary

In this chapter we have presented our Context Modelling Framework. The framework is designed to enable end users to define their own context. In order to be user friendly, expressive and extensible we have created three layers: the plug-in layer, the configuration layer and the context layer. The plug-in layer provides extensibility through the use of plug-ins that acquire external context information. The configuration layer provides expressibility through the use of templates, and the context layer provides an easy to use modelling interface for end users. We discussed how a user defined context is translated in a declarative specification that can be loaded in a rule engine for runtime evaluation. We have introduced the ExpertUser Tool and the EndUser Tool and explained how to use them.

# 5 Implementation

This chapter covers the implementation details of the ExpertUser Tool and EndUser Tool, and the integration with the Drools runtime environment. We discuss the main classes and how they cooperate.

## 5.1 Technologies and Libraries

The framework is written in Java 1.8. All graphical elements are based on JavaFX, the current standard GUI library for Java. JavaFX is included in the JDK as from version 7. The design of most graphical elements was prepared with the JavaFX scene builder. This tool allows designers to compose a scene by dragging graphical elements to their desired location. It generates an fxml file which is an xml dialect for JavaFX that can be opened by a JavaFX application to set an initial scene. When the ExpertUser Tool and EndUser Tool initially open they present an empty scene that was designed with JavaFX scene builder. Other graphical elements such as objects in the workbench are added by calling constructors at runtime.

For the persistence of objects, json-io version 2.5.1 is used[1]. This library provides a straightforward way to persist an object in a file, and initialise an object by reading it from file. The files have a JSON format.

[1] https://github.com/jdereg/json-io

## 5.2 Application Structure

### 5.2.1 Graphical Elements

This section describes the general structure of the two JavaFX applications ExpertUser Tool and EndUser Tool. The main class for a JavaFX application has to extend the *javafx.application.Application* class, and overwrite a method *start* that is called when the application is started. This method first creates the initial view from an fxml file. It then instantiates the appropriate controller object (ExpertApplicationController for the ExpertUser Tool and UserApplicationController for the EndUser Tool) and immediately passes control to it. The controller is the central class in both applications. It implements all actions initiated in the menu items, controls the workbench and all graphical object lists used by the applications. Controlling these objects is done by implementing listeners that respond to events (e.g. *on drag detected*, *on drag dropped*, etc.).

Objects in the workbench are added as children of the *AnchorPane* object that implements the workbench. These objects also create listeners that respond to relevant events (e.g. an *on drag dropped event* on an input connector of a template). To move objects in the workbench, both applications have implemented a *MakeContentDraggable* class that calculates the new position of dragged objects.

### 5.2.2 Extensible Types

As already mentioned in section 4.1.1, the framework does not predefine data types for any context information item. Since it is designed to be as flexible as possible it supports extensible data types. These data types can be defined by a plug-in developer (to create dynamic events or static data objects) or by an expert user (to create place holders for input events in templates, or explicitly specify output events in templates). The framework uses a number of classes to implement these types. Figure 5.1 gives an overview of these classes. In its most general form, a data type is an object with a name and a set of named attributes. The class *Type* is an abstract class that implements these basic properties. The abstract class can be implemented by one these three classes:

- *Event:* An *event* represents an object that can be loaded as a fact in the rule engine. An event can be defined by an end user as output of a template (indicated as a *UserDefined* event in Figure 5.1), or it can be defined by the plug-in developer as something that can be detected by a plug-in (indicated as *DynamicEvent* in Figure 5.1)(e.g. a person enters

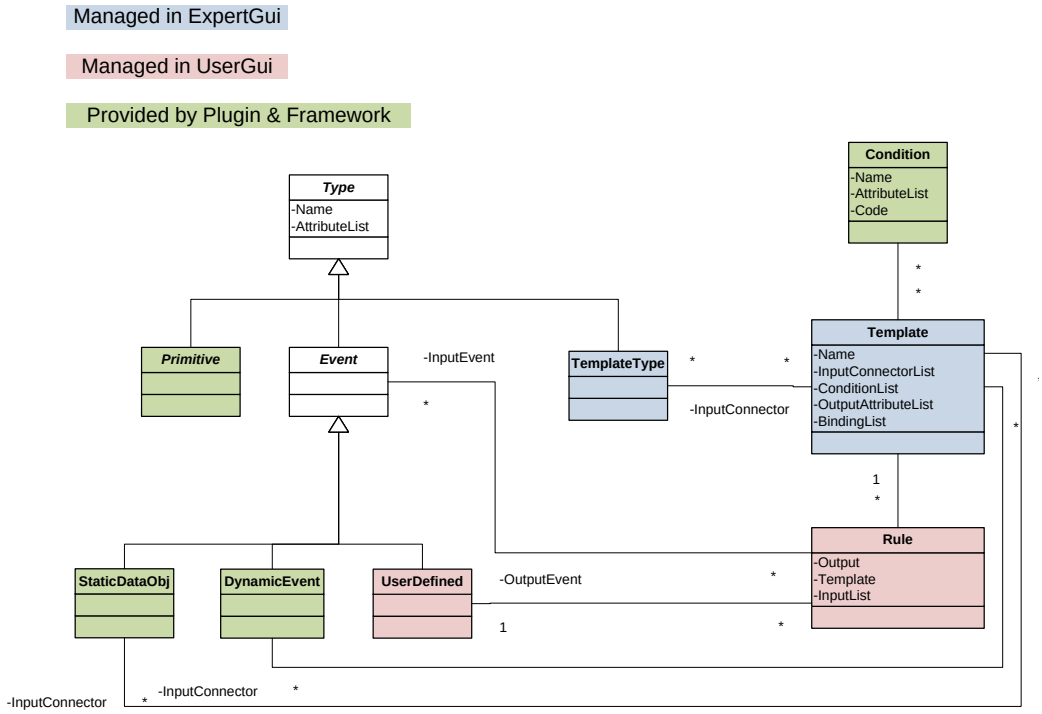


Figure 5.1: Classes that use and implement event types

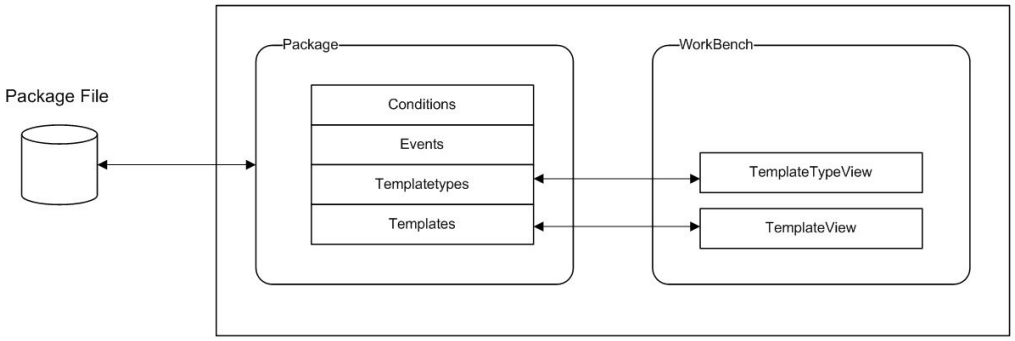


Figure 5.2: Objects managed by the ExpertApplicationController class

a location). Static data objects are also implemented by the event class since they represent data that can be loaded as facts in the rule engine. For every defined event the framework will create a declarative type definition as part of the declarative specification of an orchestration.

- *Primitive:* A *primitive* is a type that is known by the rule engine used in the runtime environment of the framework (e.g. integer, string, etc.). Primitive types have to be provided to the ExpertUser Tool and the EndUser Tool as a list. The framework will not create declarative type definitions for primitive types.

- *TemplateType:* A *TemplateType* is not used by the runtime environment. It is only used in the definition of templates. The framework will not create a declarative type definition TemplateTypes, since in an orchestration they are replaced with events.

## 5.3 The ExpertUser Tool

Figure 5.2 shows the most important components of the ExpertUser Tool. The state of the application is maintained in a singleton object from class *Package*. A *Package* contains lists for following objects :

- *Conditions:* The state contains a definition of every condition. This is essentially a description for the signature of the boolean function that implements the condition. To describe the signature we keep a list of its input attributes. The actual implementation of the boolean function is not kept here.

- *Events:* The state contains the definition of every dynamic event the

plug-in can detect. Information about static data objects (e.g. rooms, documents, etc.) is also kept in the events list.

- *TemplateTypes:* User defined types that can be used when constructing templates. These types are not used by the plug-in or the runtime evaluation process.

- *Templates:* A template combines input elements with conditions and can specify an output type. Defined in the ExpertUser Tool.

At start up, the application needs information about *primitive types* and information to instantiate its *package object*. The primitive types are passed to the controller as a list. A package object is usually instantiated in the controller by reading it from a *package file* in JSON format. However, it is also possible to create the package object explicitly using a constructor in an initialisation method and pass this object to the controller. The latter is useful when a new package is created and a file is not yet available for this package. In such case an initialisation method can create the conditions, static data objects and dynamic events and add them to the package object. Using an initialisation method avoids having to create the package file in JSON format manually, which is a time consuming and error-prone task. Remark that a new package does not contain *Templates* or *TemplateTypes* yet, these are added by the expert user.

When the expert user starts using the tool, all *TemplateTypes* or *Templates* created by the expert user are added to the main *package object*. The *package object* can be saved to a *package file* using the json io library. This library is also used to load a *package file* and instantiate a *package object* in the ExpertUser Tool.

When a new package object is created or a saved package is loaded, the four lists at the left and right side of the central workbench are populated but the workbench remains empty. To create a graphical object in the workbench, a user has to complete a drag and drop operation from one of these lists to the workbench. Every graphical object in the workbench is linked to an object that belongs to the package object (e.g a graphical *TemplateView* object has a reference to a *Template* object which belongs to a package), but exists as a separate entity as illustrated in Figure 5.2. The same *Template* can be dragged multiple times to the workbench. This will instantiate multiple new graphical objects all linked to the same template object. When an expert user manipulates a template in the workbench, only the manipulated graphical object (i.e. TemplateView) is updated. The user has to click the 'save' button in the graphical object to synchronise this graphical

object to its linked package object. Objects in the workbench are considered temporary objects that are lost when the application is closed. Only by synchronising them to the linked package object and saving that to file, changes can be persisted. Graphical objects in the workbench are composed as a hierarchy of other graphical objects. The graphical position of child elements in the AnchorPane that implements the workbench are relative to the parent element, so that the entire object can be moved around as a single entity. Figure 5.3 shows the structure of the *TemplateView* class. It basically consists of a header, a central component for the conditions, input events and the output event, and a component for the three areas where objects can be dragged to. The *TemplateConditionView*, *TemplateInputView* and *TemplateOutputView* classes shown at the bottom of Figure 5.3 again have a similar tree structure. The *TemplateInputView* for example contains a set of *TemplateInputAttributeView* objects to represent the attributes of an input event. Every element of an object tree contains a pointer to its parent, all the way up to the controller object. This enables every part of a graphical element to retrieve data from any other object.

## 5.4 The EndUser Tool

The implementation of the EndUser Tool follows a similar approach as the ExpertUser Tool, but is more complicated. The EndUser Tool loads multiple *package files* and accumulates this information in a *model* object, as illustrated in Figure 5.4. This *model object* can be used to create *orchestration objects*. When a new *orchestration object* is created from a model object, the lists in the application that present package information are populated, but the workbench is empty. Same as with the ExpertUser Tool, the workbench can be populated by completing a drag and drop operation from an object of the appropriate list to the workbench. Unlike the situation with the ExpertUser Tool however, objects in the workbench of the EndUser Tool are not considered temporary objects that are always lost when the application is closed. They are permanent elements of the orchestration and are saved to file when the orchestration is saved. The orchestration object contains all necessary information about objects in the workbench. Orchestration objects can be written to file and loaded from file. When an orchestration object is initialised by loading it from file, the workbench is immediately shown in the same state as it was when the orchestration file was saved, with all graphical elements in the workbench at the same location. However, because the json-io library can not serialise the JavaFX objects (extensions of VBox, HBox etc.) that implement the graphical objects in the workbench,

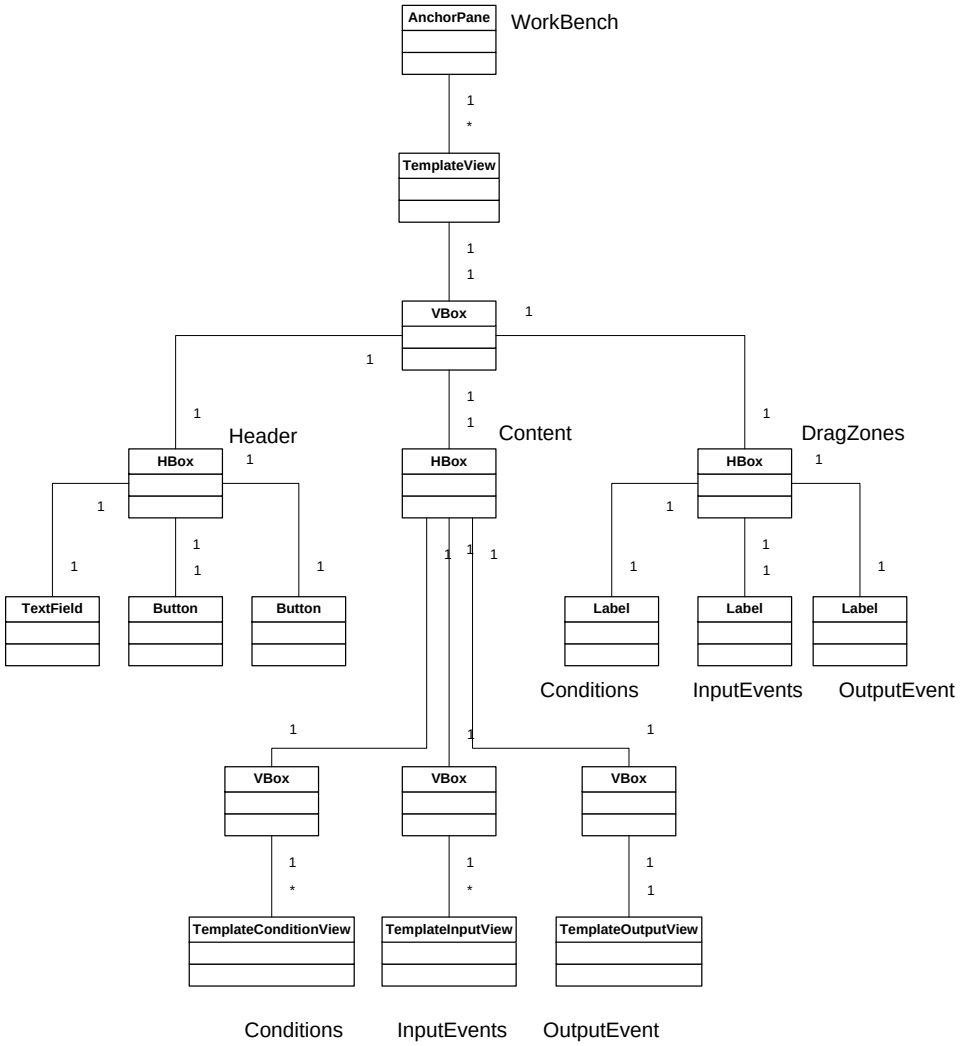


Figure 5.3: Structure of the TemplateView class

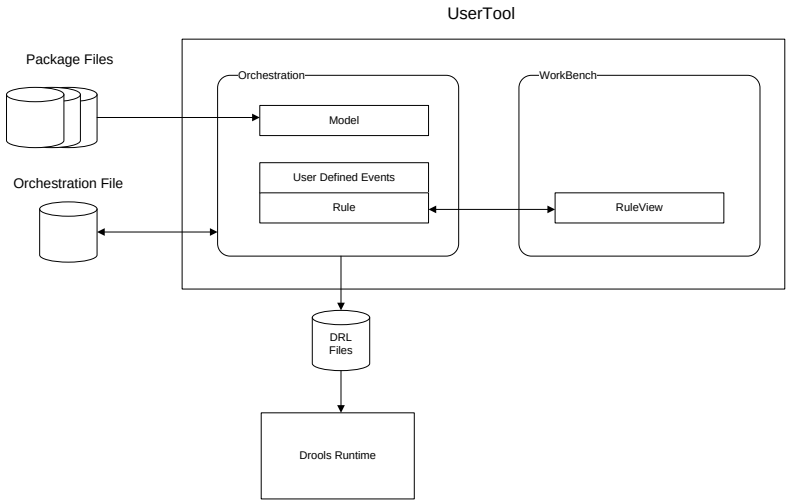


Figure 5.4: High Level Overview of the EndUser Tool

an orchestration object is created as POJO for every graphical object. These orchestration objects contain all data necessary to instantiate its corresponding graphical object. As illustrated in Figure 5.5, an Orchestration consists of a list of groups called RuleSequence. Every group contains a series of *Rule* objects. *Rule* objects correspond to templates in the workbench. A template in the workbench is implemented by the *RuleView* class.

## 5.5 Rule Generation

The EndUser Tool provides a function to translate an *orchestration object* in a declarative language. This translation results in a set of rule files and type definition files, that can be loaded in a Drools rule engine. The translation process first generates the type definition files. For every defined event, a drl file is created that defines its data type. As explained before in this chapter an event is either a dynamic event, a user defined event or a semi static object. Type definition files are generated in order of dependency : if a data type *Room* has an attribute of data type *Location*, the drl file for *Location* is generated before the drl file for data type *Room*. This dependency enables incremental loading of data types and rules in the rule engine. Incremental loading makes it possible to design an engine to which small changes in the context model can be pushed without restarting the engine. As a result, the tool can be used in complex systems which can't be restarted for every small

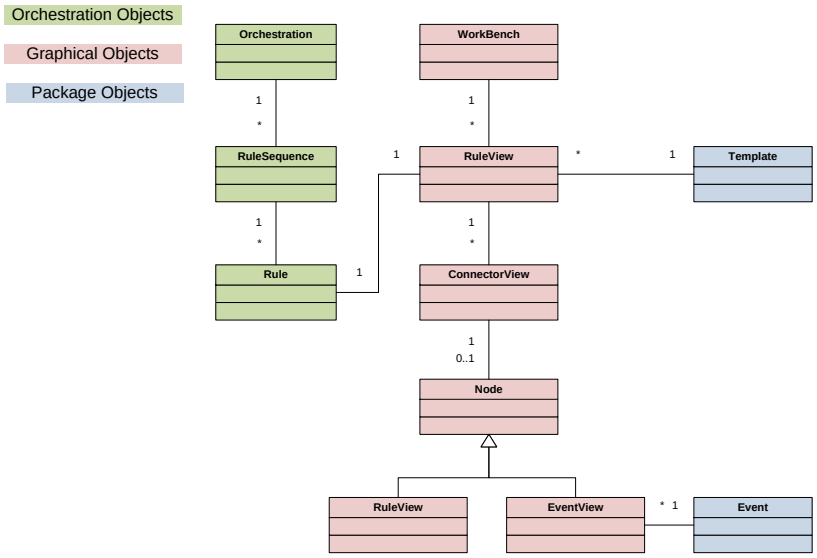


Figure 5.5: Objects in an Orchestration and their corresponding graphical objects

change (e.g. home automation). After generation of the data types, a drl file for every rule object is created (see Figure 5.6). To generate a drl file for a rule, the process follows these steps:

- It first reads the conditions of the associated template. For every condition it prepares a clause to be added to the *when* clause of the rule.
- Next, the input events are read from the template. For the input events defined with place holders, the actual linked event is retrieved from the orchestration.
- Every input event generates a conditional element in the *when* clause. The condition clauses prepared in the first step are added to these conditional elements.
- The *then* part of the rule is created. This consists of a statement to create an object of the output type, statements to set the attributes of this object to the attribute values of the corresponding input event, and a statement inserting the new object in the rule engine.

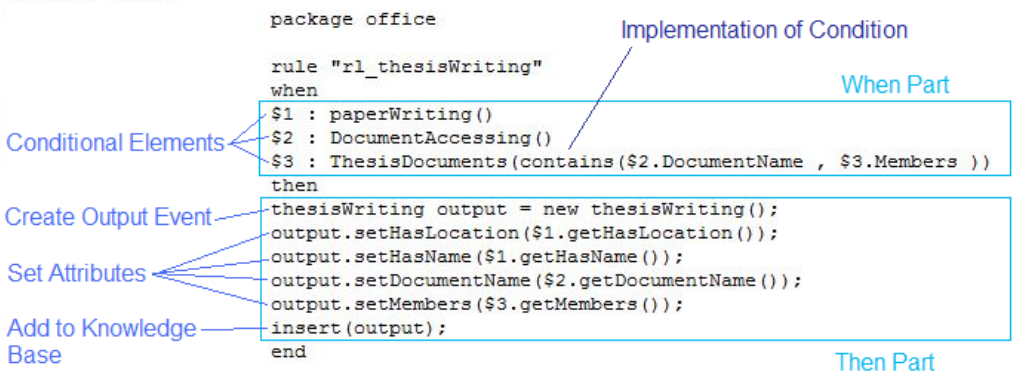


Figure 5.6: A generated Rule

## 5.6 The Rule Engine

We have used Drools [2] as a rule engine. Drools is provided by the Jboss Community and is available as an open source project. The current version of the Drools rule engine is built around the Phreak algorithm, based on Leaps and Rete/UL [32]. We did not integrate the rule engine with real sensors, but simulated a set of dynamic events and added these to the rule-engine to validate correctness of the generated rule sets.

We used version 6.0.0 of the Drools rule engine to validate the generated drl code. This was done by loading all drl files to a knowledge base, and have a Java program add facts to the knowledge base to trigger the appropriate rules. To have a Java program add a fact to the knowledge base, first the data type of the fact needs to be declared by reading in a type description from the knowledge base. Then the Java program instantiates an object of this data type and sets its attributes to the desired values. These objects can then be added to the knowledge base, after which the Java program requests the engine to fire its rules.

Remark that we do not have to share libraries or other compiled objects between the rule engine and other components of the framework. The rule engine only needs the drl files generated by the EndUser Tool, and the drl files that implement the conditions used in the templates. The drl files for these conditions need to be provided by the plug-in developer, who needs to implement the conditions as boolean functions in a format that is compatible with the rule engine.

The runtime environment of the plug-ins acquires context information. This can be implemented by connecting the plug-in to a sensor or any other

[2] https://www.jboss.org/drools/

data source. When a context information item is detected by the plug-in, it creates an event that is added as a fact to the rule engine. The rule engine is then asked to evaluate and trigger its rules.

## 5.7 Summary

In this chapter we have explained how the framework was implemented. The ExpertUser Tool and the EndUser Tool are created with JavaFX. We explained how these applications are organised and how they create their deliverables. The ExpertUser Tool creates packages, the EndUser Tool creates orchestrations. The runtime environment is a Drools rule engine. Orchestrations are translated in declarative specifications that are loaded in this rule engine. When a plug-in detects a dynamic event, it is added as a fact to the rule engine.

# 6 A Use Case in PIM

This chapters presents a use case that applies our framework to a context-aware desktop application. This context-aware desktop application was developed as a tool for the organisation and retrieval of personal information items. We start with an overview of *Personal Information Management*. We then point out the significance of context for this domain, and use our framework to model a context related to this domain. Finally we integrate the context-aware desktop application with the runtime environment of our framework and apply it to the created context model.

## 6.1 Personal Information Management

The interaction between people and their information items is being investigated in the *Personal Information Management* (PIM) research field. PIM includes both descriptive research regarding human information behaviour and the design of systems that support people in personal information related activities. PIM interacts with various research domains including cognitive psychology, human-computer interaction, artificial intelligence, information and knowledge management, information retrieval and information science. A general accepted formal definition of PIM is the one given by Jones [40]:

*"Personal information management or PIM is both the practice*

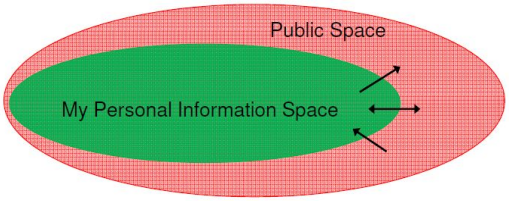


Figure 6.1: Public and Private Information Space

> *and the study of the activities people perform to acquire, organise, maintain, retrieve, use, and control the distribution of information items such as documents (paper- based and digital), Web pages, and email messages for everyday use to complete tasks (work related and not) and to fulfil a person's various roles (as parent, employee, friend or member of a community)."*

People keep information items when they satisfy a current or future information need. Storing items for future use implies that we need to be able to re-find items in the personal information space. In order to efficiently re-find previously stored items, these need to be organised in a proper way. It is generally accepted that keeping, organising and re-finding are the three main activities of PIM. They are presented by Jones [41] in the *Keeping, Organising and Re-finding* theory.

**Keeping** When people encounter information items in the public information space, they assess its value for personal use. Information that has sufficient value is added to the personal information space (see Fig 6.1). Remark that information spaces contain both digital as physical information.

**Organising** Information items that we decide to keep need to be stored. In order to efficiently re-find these stored items they need to be organised. Users organise information by applying multiple strategies. Malone was one of the first researchers who addressed the topic of organising information items in physical workplaces. In his work, he identified two strategies namely *piling* and *filing* [48]. Filing is a strategy where items need to be labelled and added to an ordered storage container. These files can again be integrated in larger files. People have applied filing strategies for centuries (e.g. a file-cabinet). The filing activity is far from trivial. To ease later retrieval, users will typically include as much information as possible in the labels. This behaviour leads to unclear and fuzzy labels. This problem is referred to as the *classification* problem. Secondly, the items have to be included in the

appropriate ordered structure (e.g. a physician adds a patient's file in a file-cabinet, alphabetically ordered by name). Since the filing activity requires time and effort, users will only spend this effort if the benefit justifies this investment [25]. Malone's research covers only the physical information space but his findings are later integrated in research on the organisation of the digital information space as well [66, 14].

**Re-finding** Re-finding information items applies to information items that have previously been kept and organised by the same user. The re-finding activity occurs in a user's personal information space, as opposed to the finding activity which applies to public information spaces. Whereas a search engine is generally used for finding activities, it is not the preferred tool for the re-finding activity. Bergman even indicates that search engines are only used as a last resort, when the location of a file is forgotten [12]. Teevan introduces another re-find strategy named orienteering [55]. Orienteering implies that people start a search at a certain point in the organisational structure and navigate step by step through the structure until the item is found. Barreau [8] and Bergman [12] have also indicated orienteering as the main strategy for re-finding of information. A possible explanation for the popularity of orienteering is the preference of recognition over recall. A search engine requires formulating one or several keywords that need to be recalled from the semantic memory. The orienteering strategy uses only recall by selecting one of several options when taking the next step in the navigation process. Another advantage of orienteering is that the folder hierarchy can contain additional context information which helps the user in the re-finding process.

## 6.2 The Human Memory

In order to address the relevance of context for PIM, we first need to explain some concepts about human memory. It is commonly accepted that the human memory does not work as a single solid system but consists of several subsystems. Atkinson and Shiffrin introduced a psychological model that consists of three interdependent layers namely the sensory memory, short-term memory and long-term memory [3]. The sensory memory is directly related to the human input channels (i.e. sight, hearing, touch, smell, taste and balance) and acts as a buffer between these senses and the rest of our cognitive system. Most of the information gathered by our senses is not considered and therefore quickly lost. An example can be a situation where person is talking to a second person who is not actually listening. The person

spoken to is not able to repeat any of the words. By paying attention to one of the stimuli, information is transferred from the sensory memory to the short-term memory. In our example this would mean that the second person starts to listen actively. In short-term memory, the information items are stored for an average of about 30 seconds, unless they are consciously repeated. By explicitly processing (i.e. encoding) the information, it is transferred to the long-term memory where it becomes available for retrieval. Squire [52] extended the previous cognitive model by partitioning the long term memory in two subsystems namely the declarative memory and the non-declarative memory. The non-declarative memory manages among others procedural memory (e.g. riding a bike) and associative learning (e.g. conditioning). The declarative memory system is responsible for the storage and processing of events and facts.

Tulving [59] further divides the declarative memory system in the *episodic* and *semantic* memory. The episodic memory receives and stores personal experiences. It can be pictured as a sequential list of events. Every time a person perceives something, an event is added to this list. The episodic memory stores the perceptible attributes of events and their temporal-spatial relations. Events are always stored in terms of their autobiographical references to the already existing events in episodic memory, and to the previous event. The episodic memory has a very high transformation speed since every experience adds a new event. Even a retrieval action from episodic or semantic memory creates an event that will be added to the episodic memory. On the other hand, the semantic memory consists of facts and concepts that we have learned. Instead of the sequential structure from the episodic memory, it can be pictured as a semantic graph of concepts that are connected through associative links. The semantic memory can be updated by receiving new input (e.g. learning new concepts by reading a book) or by reasoning on the existing semantic graph (e.g. deducing new concepts from existing ones). Compared to the episodic memory, the semantic memory transforms slower but it is much more complex from structure.

For most cognitive actions, both episodic and semantic memory systems are involved. Let us illustrate this interplay of both systems by an example of a student attending a course (illustrated by event E2 in Figure 6.2). The course takes place at 09:00 am on a Monday morning in classroom 10F720. Furthermore, the lecturer's topic concerns multi-modal interaction. This event will be stored in the student's episodic memory together with the perceptible attributes such as the classroom, time, other students present and even the fact that the student is hungry since there was no time for breakfast. This event will have an auto-biographical reference to the event of having a

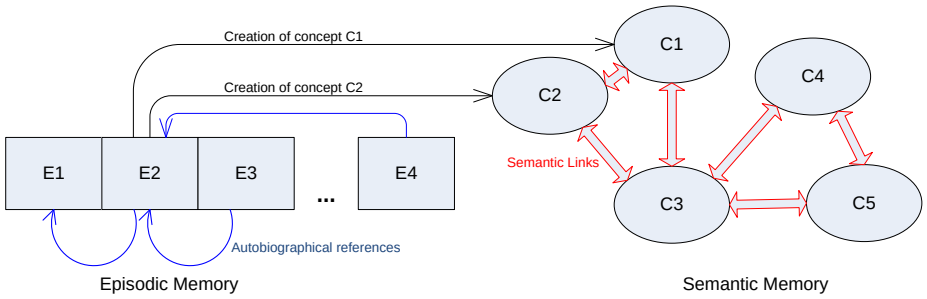


Figure 6.2: Interaction between episodic and semantic memory systems

conversation with other students just before class (illustrated by event E1 in Figure 6.2). The learned concepts on multi-modal interaction gained in the lecture will be stored in the semantic memory (concepts C1 and C2 in Figure 6.2). When studying the material (concepts C1 and C2 in Figure 6.2) for the exam later (event E4 in Figure 6.2), the student will recall the event of attending the class where this material was explained. As illustrated by this example, semantic and episodic memories are interdependent. A retrieval from the semantic memory updates the episodic memory. Even the encoding of perceptual events in episodic memory can be influenced by concepts in semantic memory. The interaction between semantic and episodic memory receives a lot of attention in the research field of cognitive psychology [16].

The structure of these memory systems has major consequences for the retrieval of information. For information to be retrieved, it needs to be accessed by following the available references and associative links. The auto-biographical references in the episodic memory play an important role in retrieval actions. Together with the perceptible properties of events, these references are used to answer queries like

- what did I do at 10:00 am this morning
- what did I do next
- what activities are related to this location
- what are the items on my shopping list

The information provided by auto-biographical links and perceptible properties of events stored in episodic memory is additional information to an event. Applying our definition of context given at section 2.1, we can conclude that this information is indeed context of an event.

## 6.3 Context applied to PIM

Time cues, spatial cues and contextual cues are important for the re-finding activity. Since semantic memory stores perceptible attributes of events including time, location and context, it is natural human behaviour to use these cues. The use of cues in different organising strategies has been investigated in a survey by Trullemans and Signer [57]. As shown in Table 6.1 contextual cues are used for all identified strategies both in physical as in digital information spaces. The importance of context for the keeping and re-finding activities for digital file-systems had also been addressed by Barreau [8].

| | | Context cue | Spatial cue | Time cue |
|---|---|---|---|---|
| Physical space | Piling | ✓ | ✓ | |
| | Mixing | ✓ | | ✓ |
| | Filing | ✓ | ✓ | ✓ |
| Digital space | Piling | ✓ | | |
| | Mixing | ✓ | | ✓ |
| | Filing | ✓ | | |

Table 6.1: Cues supporting the re-finding activity [57]

Though contextual cues are important, existing organisational strategies do not directly support context. Lack of support for context is one of the main problems in current information management systems. According to research done by Malone [48] the classification problem originates from the difficulty of setting up an efficient organisation structure. Providing the item with appropriate labels causes again a cognitive overload [42]. The root cause of these issues is the fact that most organisation structures supported by current information systems are fundamentally different from the structure of human memory. This applies to information systems both in the physical space as in the digital space. As already mentioned by Bush [20] in 1945 in his paper *As We May Think*, the human mind does not organise information according to hierarchical structures (as in digital or physical file-systems), but it links information items using associations. This vision corresponds with the structure of semantic memory introduced in the previous section. However, it doesn't recognise the important role of context in re-finding activities. User activities occur in a context, meaning that the activity contains contextual factors. For example, a meeting takes place in the context of a project. Information items related to this activity (e.g. the meeting minutes) refer to the same context. If a user decides to keep an information item, this keeping activity takes place in a context (e.g. the project). When

storing the information item, a user typically tries to include context information (e.g. the project). When filing strategies are applied, the context information can be encoded in the file hierarchy by using dedicated folders (e.g. named after the project). The re-finding activity also takes place in the context (e.g. the user might search in folders that are named according to the project). In this case, next to the project context factor, additional context information can be used, such as the date of the meeting.

To model the role of context in retrieval from human memory, Trullemans and Signer [58] proposes a conceptual human memory model consisting of three layers (shown in Figure 6.3):

- The object level : Objects are real world elements, either from the physical or the digital world. They can refer to information items or parts of information items. Objects can be connected through navigational (e.g. hyper-links) or structural (indicating composition) links.

- The concept level : concepts are general ideas, that only exist in the mind. They abstract the complexity of the world. A concept can for example be the label a user gives to a folder. Concepts can have associative links, as explained in Quillian theory of semantic memory. Objects and concepts can be related by extent links, indicating a categorical relationship between object and concept

- The contextual level : a context is a composition of contextual factors. In accordance with the definition of context given by Dervin [Dervin, 1997], contextual factors are conditions or observations that make objects or concepts more understandable. A context can have links to objects, concepts, and their links. These links can have individual weights, indicating the relevance of the object or concept for a context.

The selection of a context element will activate linked elements at object and concept level. Activated elements can in turn activate other linked elements until a certain depth, resulting in the selection of a sub-graph of objects and concepts. This sub-graph contains the items relevant in a given context. By explicitly modelling contextual links, this model directly supports re-finding based on context cues.

A good PIM system should allow users to retrieve information in a similar way as how they retrieve information from their own memory. Since PIM activities depend on context, we believe that a PIM system should include direct support for context. Instead of being stuck with existing organisational strategies such as Filing, new context based strategies can be developed.

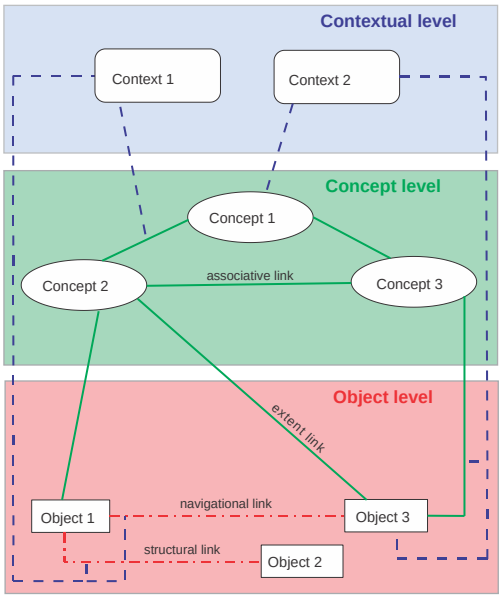


Figure 6.3: Conceptual Model for the role of context [58]

## 6.4 A User-defined Context for Information Items

### 6.4.1 A Context Example

As mentioned in above section, information items are related to a context, and a good PIM system should be able to keep this context information when storing the information item. As discussed in section 2.2 the user should be involved in the definition of these contexts. To illustrate this idea we will now use our modelling framework to define a context for personal information items. As a running example in this section we use the `Working on my Thesis` context that applies to all information items used to compose this thesis. In this example we use the following types of atomic context information to derive higher level context information:

- My calendar and agenda : the current agenda can indicate a timeslot as being working time, free time or in meeting.
- My location
- My activity: I can be browsing for information (physical or digital), typing or talking.
- The information item a I am accessing

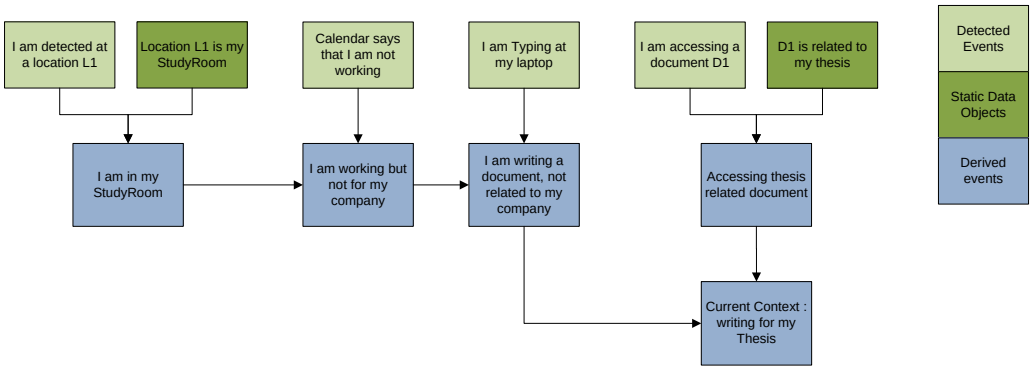


Figure 6.4: Sample derivation for `working on my thesis` context

Suppose that the application has also access to a set of static data objects such as relevant places with their location (e.g. the study room or meeting rooms in an office), and knows what information items related to certain domains. We then define the context `Working on my thesis` as shown in Figure 6.4: I am working on my thesis if I am in my study room while not during working hours (to exclude the possibility that I would be working for my company), typing on my laptop and accessing documents the systems knows to be related to my thesis. Remark that this scenario is only one way to define the `working on my thesis` context, it can co-exist with other scenarios that together define this context.

### 6.4.2 Creating Templates

We now explain how this example can be implemented in our modelling framework. For the four types of atomic context information, plug-ins would have to be created. These plug-ins need to have a specification that can be used by the modelling tools and a runtime component that can deliver the detected events as facts to the rule-engine. For every plug-in we have created a specification in a package :

- Calendar: A package that can access a users calendar and the status of the current timeslot. In our implementation it only differentiates between `working hours` and `non working hours` but various other types could be defined (e.g. holiday, in meeting, non working hours etc.). When the calendar status changes it creates a dynamic event.

- Location: A package that tracks a users location. When a user is tracked at a new location it creates a dynamic event. It also has static information about rooms and places. The package includes a condition `inRange` that checks whether two locations are within a certain range from each other.

- Activity: A package that tracks a users current activity. When it detects a new activity it creates a dynamic event.

- Documents: A package that tracks a recently accessed information items. When it detects an item being accessed it creates a dynamic event. The packages has static data about sets of items that are relevant for a domain. The package has a condition `isMember` to check whether a value of String belongs to a list of values of type String.

When the `Documents` and `Location` packages are loaded in the Expert Tool we can create the required templates. Since this tool can only work on one package at the same time this is done in separate steps. For the `Documents` package we add a template `Wile Accessing Document of`. Figure 6.5 shows the workbench of the ExpertUser Tool while creating this template. The template has three input events:

1. An empty place holder to which anything can be applied. Templates raise the level of abstraction by combining information. The `While Accessing Document of` template combines any generic information item with the fact that at the same time a document is being accessed. This place holder is used to represent this generic information item. A user defined type `empty` was created for this place holder. This type has no attributes. Since an empty set is a subset of any set, every data type is compatible with this place holder in the sense of the definition given in section 4.1.1.

2. The dynamic input event `Document Accessing` that the plug-in can detect.

3. A place holder for an information item with a data type that has at least an attribute with name `Members` and type `List`. This place holder can be used to specify a list of documents known to be relevant for a domain. A user defined type `hasList` was created for this place holder. This type has a single attribute with name `Members` an type `List`.

The condition `isMember` is used in this template to validate that the information item detected by the plug-in belongs to the list of items applied to the

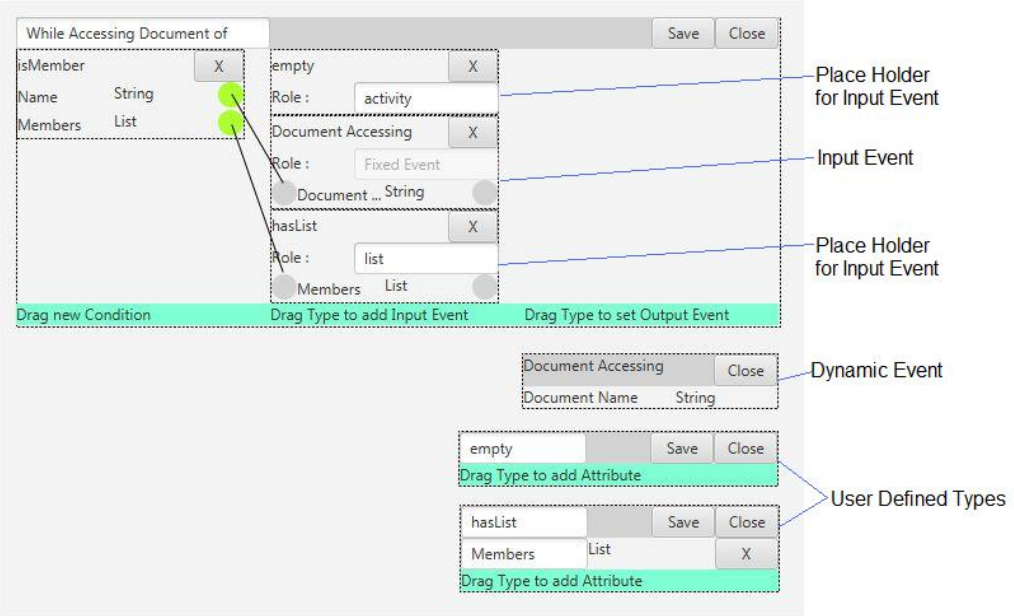


Figure 6.5: Workbench with the `While Accessing Document of` template

last place holder. The template does not define the structure of the output event. This implies that all attributes from the information item applied to the first place holder are automatically copied to the output event. If this information item does not have an attribute with name `Document Name` yet, the name of the actual document detected by the plug-in is also copied to the output event. Finally, all attributes from the information item applied to the last place holder are copied to the output event, as long as an attribute with that name was not yet added. By defining the template like this, we can summarise that it enriches any activity with information about a document that is accessed during this activity. The activity is *enriched* with information about an accessed document.

In the `Location` package a template `Nearby` is created. This template has two place holders for input information items with a location. For these place holders a user defined data type `hasLoc` was created. This data type has a single attribute with name `Has Location` and type `Location`. Any information item that has such an attribute can be applied to the place holders. The template applies the `inRange` condition to the locations of these two information items. Same as with the `While Accessing Document of` template, the structure of the output event is not defined in the template. Figure 6.6 shows the workbench of the ExpertUser Tool while creating this template.

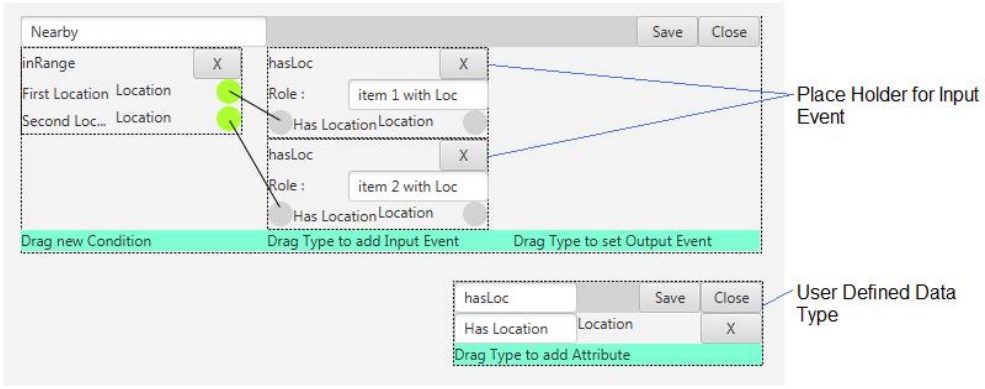


Figure 6.6: Workbench with the `Nearby` template

Finally a generic template `At Same Time` is created with two place holders and no conditions. For the place holders a user defined data type `empty` is used that has no attributes. The template does not define the structure of the output event. This template is a generic: it combines any two information items and copies the attributes of the first information item to the output event. Attributes of the second information item are also copied to the output event as long as their attribute names did not occur yet in the first information item. For this generic template a new package `Basic` was created. This package is not related to any plug-in, it only provides this template.

### 6.4.3 Creating the Orchestration

With the templates added to the packages, we can create the orchestration that implements the scenario `Working on my Thesis` introduced in Section 6.4.1. The package files for the four packages are placed on the package directory of the EndUser Tool. When menu item 'new' is selected in the EndUser Tool it reads the information from these package files and populates the lists for templates, static data objects and dynamic events.

To implement the first step of he derivation illustrated in Figure 6.4 we need to verify if `my location` is the same location as the `studyRoom`. This is done by applying the `Nearby` template to the dynamic event `MyLocation` and the static data object `MyStudy`. These were are provided as part of the `Location` package. We first have to drag the `Nearby` template to the workbench, dropping it on the first vertical line, then drag the selected information items to the input place holders. The name of the output event is set to `inStudyRoom`, after which it appears in the list of user defined contexts. The result of these actions is shown in figure 6.7. The curved lines indicate

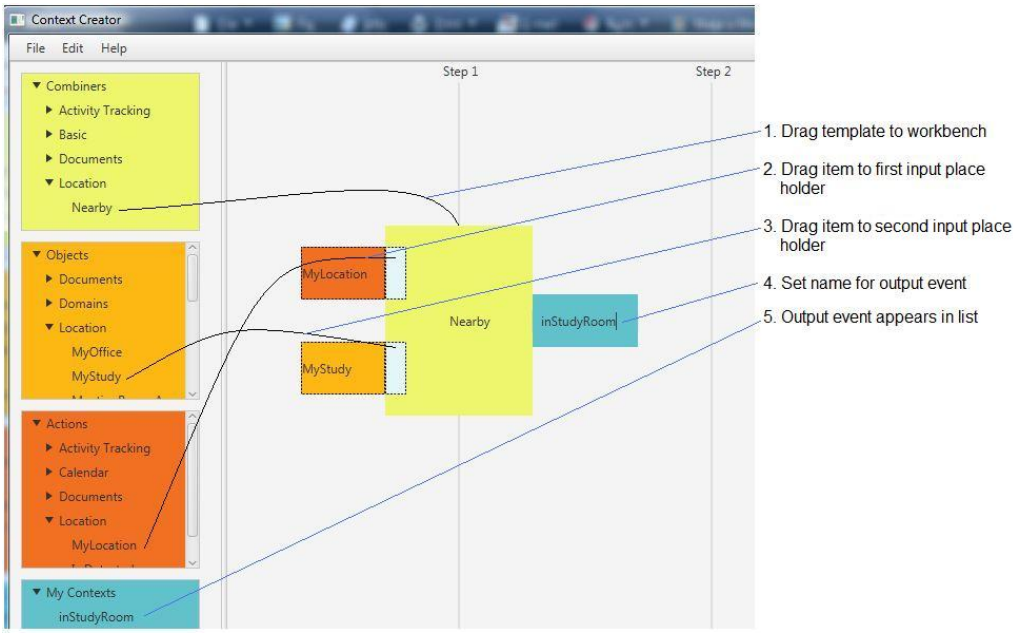


Figure 6.7: First step to derive the `Working on Thesis` context

a *drag and drop* action from an object in a list to the workbench.

To implement the second step of he derivation illustrated in Figure 6.4 we need to add the information that the users calendar indicates the current timeslot as `Not Working Time`. This information is provided as a dynamic event in the `Calendar` package. We build further on the previous step by dragging the `At Same Time` template to the workbench. The template is dropped on the `inStudyRoom` output event from the previous step, automatically assigning this output event to the first compatible input place holder of the `At Same Time` template. The dynamic event `Not Working Time` is dragged to the second input place holder. The name of the output event is set to `personalWork`. The result of these actions is shown in figure 6.8.

To implement the third step of he derivation illustrated in Figure 6.4 we need to add the information that the user is `Typing`. This information is provided as a dynamic event in the `Activity Tracking` package. We build further on the previous step by dragging the 'At same time' template to the workbench. We could have dropped this template on the `personalWork` output event from the second template. This would result in three templates being graphically connected. Instead we dropped the template on a free location at the first vertical line. This means that we have to set the user defined event `personalWork` manually to the first input place holder. The dynamic event `Typing` is set to the second input place holder. The name

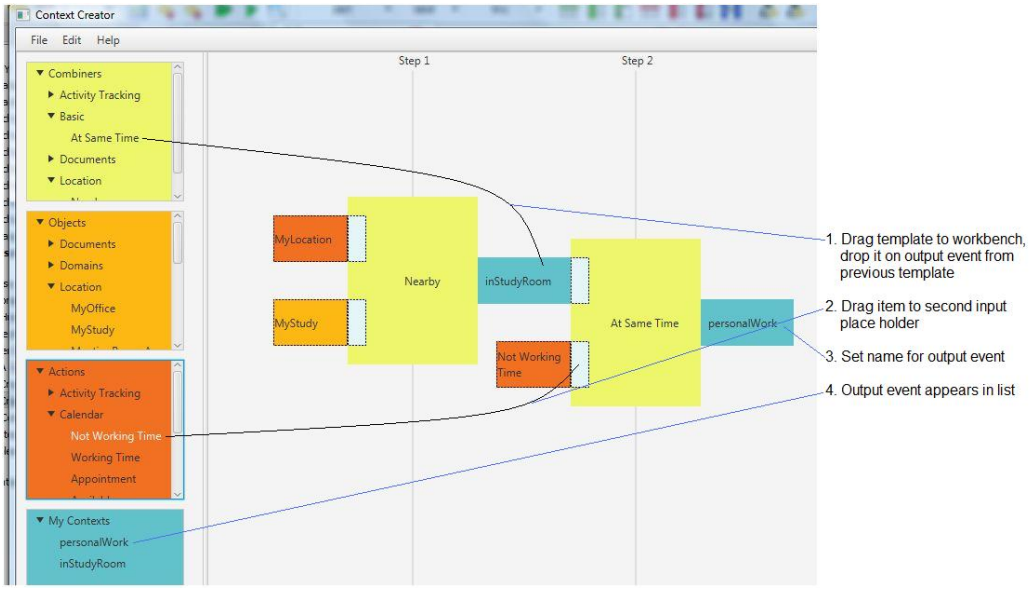


Figure 6.8: Second step to derive the `Working on Thesis` context

of the output event is set to `paperWriting`. This step is shown in figure 6.9. Remark that the `paperWriting` output event already combines the information that a user is located in the `StudyRoom`, at a timeslot in the calendar set as `Not Working Time` while the activity tracking plug-in detects a `Typing` activity.

To implement the last step of he derivation illustrated in Figure 6.4 we need add the information that the user is accessing a document related to the thesis. The template `While Accessing Document of` shown in figure 6.5 was created to add this type of information to any possible information item. We now apply this template to the `paperWriting` output event. After dropping the template on the `paperWriting` output event and selecting the second place holder, the EndUser Tool looks as in figure 6.10. Since the 'Show all' checkbox is not set, the lists for static data objects, dynamic events and user defined contexts only show items compatible with the selected input place holder. The only compatible items are the three shown static data objects in the `Documents` package. From these objects, we select the `Thesis Documents` item and drag it to the last input place holder. We then specify the name for the output event as `thesisWriting`. This concludes the modelling activities. The final orchestration is shown in figure 6.11. The EndUser Tool can now translate the orchestration in a declarative specification.

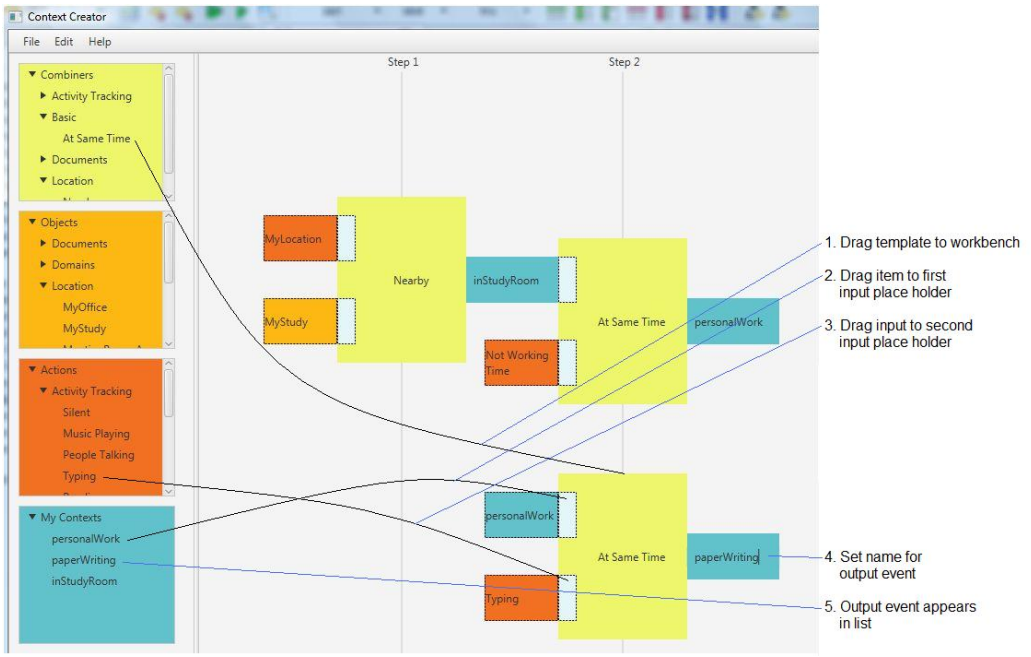


Figure 6.9: Third step to derive the `Working on Thesis` context

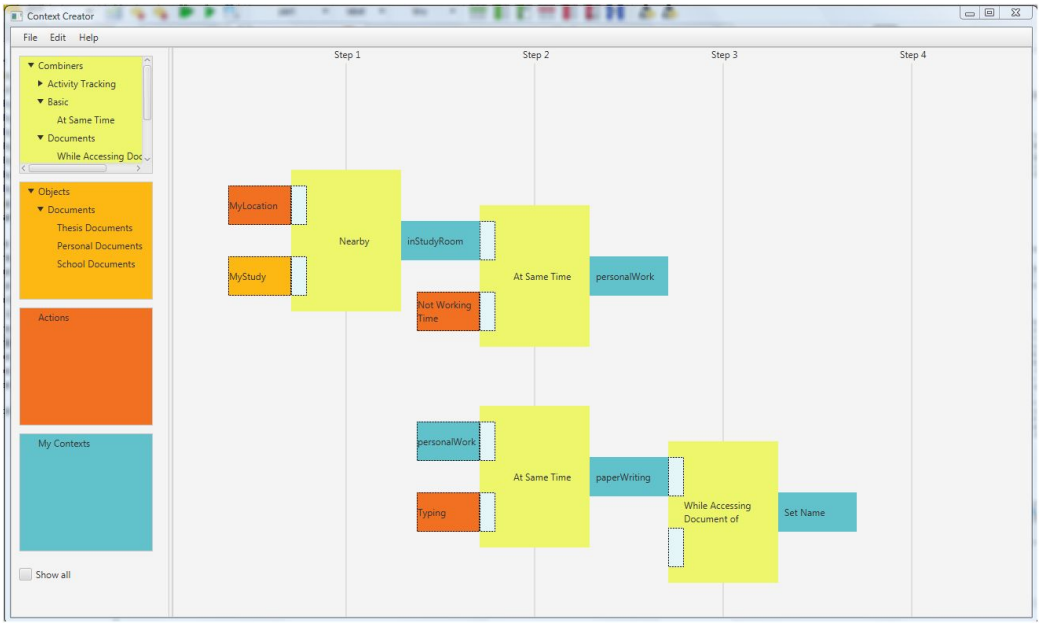


Figure 6.10: Last template added to derive the 'Working on Thesis' context

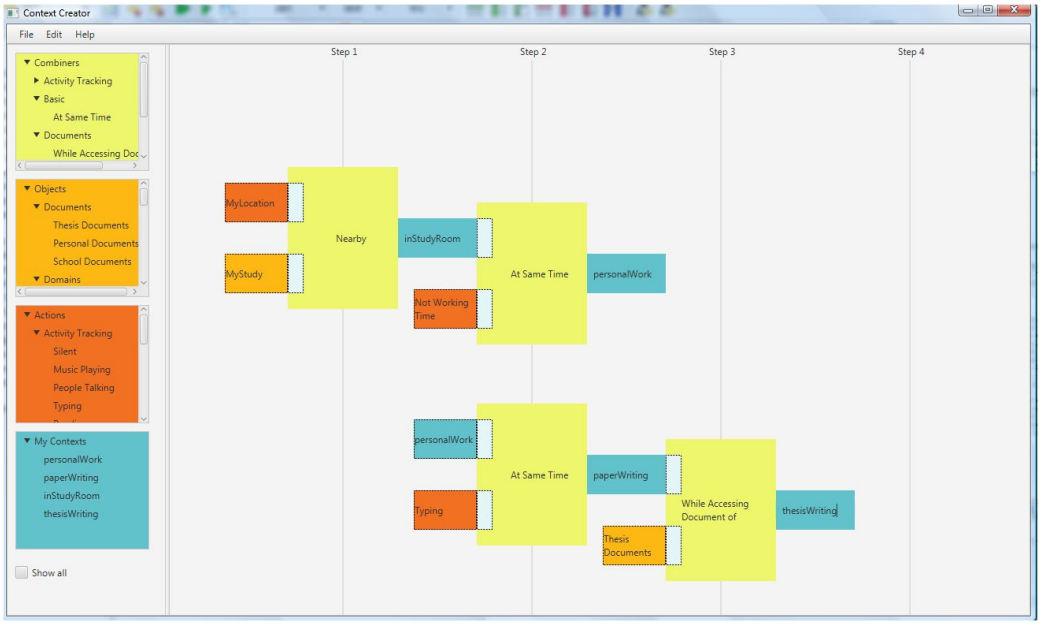


Figure 6.11: Completed orchestration for the `Working on Thesis` context

## 6.5 The Context-aware Desktop

In [56] Trullemans presents a context-aware desktop application, that shows users a different desktop depending on the current context. Every context contains its own personal information items. These desktops allow the creation of piles, and support labels and annotations that can be used in retrieval activities. In the original implementation, the current context is set manually by the user. We have integrated this context-aware desktop with the runtime environment of our framework. We added a `DroolsIntegration` component, that starts up the Drools rule engine and manages its knowledge base. The component loads the elements of translated orchestrations together with static data objects in the knowledge base, and integrates listeners for plug-ins. The component acts as a controller between plug-in devices, the rule-engine and the context-aware desktop as illustrated in figure 6.12.

As a proof of concept, we have applied this integration to the `Working on Thesis` scenario that was introduced in section 6.4.1. In the previous section we created an orchestration for this context and generated a declarative specification. We now load this specification in the knowledge base of the Drools rule engine. We added the required static data objects with information about rooms and document lists. We simulated the dynamic events `MyLocation`, `Not Working Time` and `Typing`. We have implemented a listener for the `Document` plug-in that listens to dynamic events being

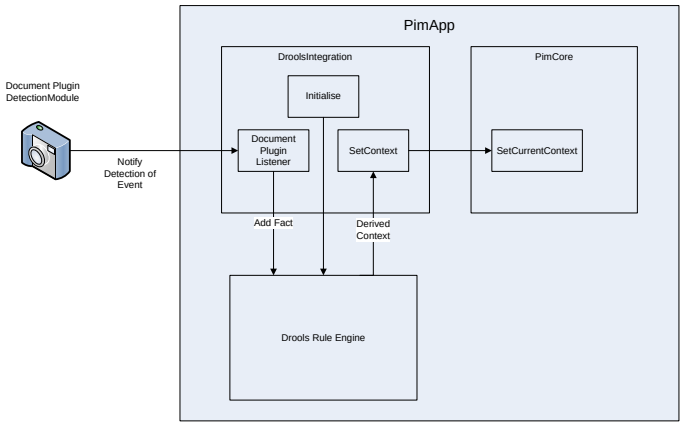


Figure 6.12: Context-Aware Desktop Integration

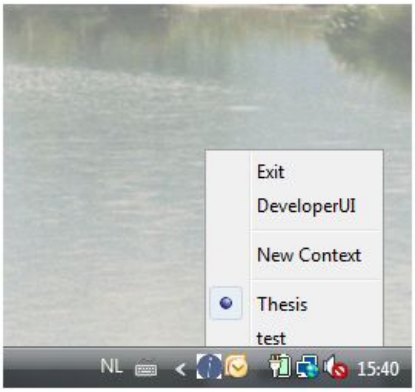


Figure 6.13: Context-Aware Desktop set to `Thesis` context

detected by a `Document Accessing Detection` module and creates fact for these events that it adds to the knowledge base. We simulated this `Document Accessing Detection` module to provide the required input events.

Upon detection of the user defined context `Working on Thesis`, the `DroolsIntegration` component activates the `Thesis` context in the context-aware desktop. Figure 6.13 shows the activated context in the context-aware desktop application.

## 6.6 Summary

In this chapter we presented a proof of concept for our framework related to the domain of PIM. We introduced the PIM domain and argued why context is a relevant factor in this domain, starting from human memory models. We then created a sample user defined context. We explained how plug-ins should be created to deliver the context information required in this sample context. We then explained what templates are required and how they can be combined to create an orchestration that models this sample context. The orchestration was translated in a declarative specification. We integrated the 'Context-Aware Desktop' with the runtime environment of our framework and loaded the generated declarative specification. We then simulated the plug-ins to deliver the atomic context information required to derive the sample context. Upon detection of the user defined sample context, this context is activated in the Context-Aware Desktop application.

# 7 Conclusions and Future Work

## 7.1 Discussion

Context-aware computing becomes increasingly important. Many frameworks already exist to support the development of context-aware applications. However, as Belloti and Edwards [9] point out, most of these frameworks do not provide sufficient user control or intelligibility. This is even more important for applications that use complex human related context. For such applications, the processing of context information cannot be deterministically modelled by a designer. The user needs to be involved in the processing of context information. User control can be provided in many ways, varying from simple personalisation (e.g. setting a specific ring-tone in a mobile phone) to complex end user programming. In order to build context-aware applications that support all forms of context while providing sufficient user control and intelligibility, end users should be able to control context models in the most generic way.

We have created a modelling framework that allows end users to define their own context models. To maximise both expressive power and end user usability, the modelling task is spread over three layers targeted to different roles: a plug-in developer, an expert user and an end user. Each step produces output deliverables that are needed in the next steps. These deliverables are implemented as configuration files that can easily be distributed

and installed. This allows end users for example to quickly pick up new relevant templates created by various expert users working on a compatible plug-in. Changing plug-ins or templates in the Expert Tool or EndUser Tool is as simple as opening a new model or orchestration file and does not even require restarting the Tools. The framework applies rule-based reasoning, but does not impose any other constraint on what can be configured. Even the primitive data types of the runtime evaluating environment are not hard coded in the modelling tools. The templates are capable of implementing business logic without exposing too much technical details to the end users. At the same time, the templates are generic in the sense that they can be applied to different events. The output events that are generated by applying templates, are not defined inside the template but by the input events an end user selects. The same template can be used to generate completely different output events. As a proof of concept we have integrated the runtime environment of our framework with a context-aware desktop application. We used the modelling tools to create a user defined context. This context was translated in a declarative specification and loaded in the runtime environment. We then simulated the context data to trigger the user defined context.

## 7.2 Future Work

Business logic is currently modelled with conditions. These are boolean functions that are evaluated by the rule engine at runtime. If these conditions evaluate to true, derived events are generated. If more general functions would be allowed, a rules could generate events containing data not explicitly present in the input events. By enabling remote function calls, the evaluation of functions can be handled outside of the rule engine. Various AI based classifiers deployed outside the framework could become accessible for the framework. Another extension could be to explicitly support the concept of probability in the modelling process (e.g. if this event occurs with probability x, create a derived event).

The framework is able to generate the executable rules in an incremental manner, in the sense that adding or changing a part of an existing orchestration will only modify the impacted part of the generated rule sets. However, in our proof of concept set-up loading these rule sets in the runtime environment of the rule engine is not yet an incremental process. All types and rules are loaded at launch of the evaluation process, after which detected events can be added to trigger activation of the rules. Changing rules at runtime is not implemented, though the Drools rule engine theoretically supports it. By enabling the modification of loaded rules without restarting the engine

(and loosing previously detected and added facts), a user can be given more control over the process: the modelling and evaluation activities can be intertwined. The integration of external data sources or sensors was simulated in a straight forward manner. In order for the runtime environment to keep an accurate state of the environment, it should not only insert facts, but also remove or update previously inserted facts. The integration of external data sources with the runtime environment should be investigated further; with possible implications for the modelling process (e.g. options for a user to specify how recent an event should be).

Our current implementation provides only limited feedback during the evaluation process. The intelligibility could be improved by providing graphical feedback using the end user modelling tool. For example, templates can change colour when a related rule is triggered in the rule engine. Various types of feedback could be investigated to improve intelligibility.

# Bibliography


[1] Gregory D Abowd, Christopher G Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A Mobile Context-Aware Tour Guide. *Wireless Networks*, 3(5):421–433, 1997.

[2] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of HUC '99, 1st International Symposium on Handheld and Ubiquitous Computing*, pages 304–307, Karlsruhe, Germany, September 1999.

[3] Richard Atkinson and Richard Shiffrin. Human Memory: A Proposed System and its Control Processes. *Psychology of Learning and Motivation*, 2(1):89–105, 1968.

[4] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[5] David Bannach, Paul Lukowicz, and Oliver Amft. Rapid Prototyping of Activity Recognition applications. *IEEE Pervasive Computing*, 7(2):22–31, 2008.

[6] Jakob E Bardram. The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. In *Proceedings of Pervasive 2005, Third International Conference on Pervasive Computing*, pages 98–115, Munich, Germany, May 2005.

[7] Louise Barkhuus and Anind Dey. Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Examined. In *Proceedings of UbiComp 2003, 5th International Conference on Ubiquitous Computing*, pages 149–156, Seattle, USA, 2003.

[8] Deborah Barreau. Context as a Factor in Personal Information Management. *Journal of the American Society for Information Science*, 46(5):327–339, 1995.

[9] Victoria Bellotti and Keith Edwards. Intelligibility and Accountability: Human Considerations in Context-Aware Systems. *Human-Computer Interaction*, 16(2-4):193–212, 2001.

[10] Victoria Bellotti and Abigail Sellen. Design for privacy in ubiquitous computing environments. In *Proceedings of ECSCW '93, European Conference on Computer Supported Cooperative Work*, Milan, Italy, September 1993.

[11] Massimo Benerecetti, Paolo Bouquet, and Chiara Ghidini. On the Dimensions of Context Dependence: Partiality, Approximation, and Perspective. In *Modeling and Using Context*, pages 59–72. Springer, 2001.

[12] Ofer Bergman, Ruth Beyth-Marom, Rafi Nachmias, Noa Gradovitch, and Steve Whittaker. Improved Search Engines and Navigation Preference in Personal Information Management. *Transactions on Information Systems*, 26(4):20, 2008.

[13] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A Survey of Context Modelling and Reasoning Techniques. *Pervasive and Mobile Computing*, 6(2):161–180, 2010.

[14] Richard Boardmann and Angela Sasse. "Stuff Goes Into the Computer and Doesn't Come Out": A Cross-Tool Study of Personal Information Management. In *Proceedings of CHI 2004, ACM Conference on Human Factors in Computing Systems*, pages 583–590, Vienna, Austria, April 2004.

[15] Nicholas Andrew Bradley. *A User-centred Design Framework for Context-aware Computing*. PhD thesis, University of Strathclyde, 2005.

[16] Nick Braisby and Angus Gellatly. *Cognitive Psychology*. Oxford University Press, 2012.

[17] Peter J Brown. The Stick-e Document: a Framework for Creating Context-Aware Applications. In *Proceedings of EP '96, Electronic Publishing*, pages 182–196, Palo Alto, USA, 1996.

[18] Peter J Brown, John D Bovey, and Xian Chen. Context-aware applications: from the laboratory to the marketplace. *Personal Communications, IEEE*, 4(5):58–64, 1997.

[19] Jay Budzik and Kristian J Hammond. User Interactions with Everyday Applications as Context for Just-in-time Information Access. In *Proceedings of IUI 2000, 5th International Conference on Intelligent User Interfaces*, pages 44–51, New Orleans, USA, January 2000.

[20] Vannevar Bush. As We May Think. *Atlantic Monthly*, 176(1):101–108, 1945.

[21] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, Hanover, USA, 2000.

[22] Guanling Chen and David Kotz. Context Aggregation and Dissemination in Ubiquitous Computing Systems. In *Proceedings WMCSA 2002, the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pages 105–114, Callicoon, USA, June 2002.

[23] Harry Lik Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, 2004.

[24] Herbert H Clark and Susan E Brennan. Grounding in Communication. *Perspectives on Socially Shared Cognition*, pages 127–149, 1991.

[25] Ian Cole. Human Aspects of Office Filing: Implications for the Electronic Office. In *Proceedings of HFES 1982, Human Factors and Ergonomics Society Annual Meeting*, volume 26, pages 59–63, Seattle, Washington, USA, October 1982.

[26] Graham M Davies and Donald M Thomson. *Memory in context: Context in memory*. John Wiley & Sons, 1988.

[27] Brenda Dervin. Given a Context by Any Other Name: Methodological Tools for Taming the Unruly Beast. In P. Vakkari, R. Savolainen, and B. Dervin, editors, *Proceedings of ISIC 1996, International Conference on Research in Information Needs, Seeking and Use in Different Contexts*, pages 13–38, Tampere, Finland, 1997. Taylor Graham.

[28] Anind K Dey. Context-Aware Computing: The CyberDesk project. In *Proceedings of AAAI 1998, Spring Symposium on Intelligent Environments*, pages 51–54, Palo Alto, USA, March 1998.

[29] Anind K Dey. Modeling and Intelligibility in Ambient Environments. *Journal of Ambient Intelligence and Smart Environments*, 1(1):57–62, 2009.

[30] Anind K Dey, Gregory D Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, 16(2):97–166, 2001.

[31] Anind K Dey and Alan Newberger. Support for Context-Aware Intelligibility and Control. In *Proceedings of CHI 2009, ACM Conference on Human Factors in Computing Systems*, pages 859–868, Boston, USA, 2009.

[32] Robert B Doorenbos. *Production Matching for Large Learning Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA, 1995.

[33] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing Search in Context: The Concept Revisited. In *Proceedings of WWW '01, 10th International Conference on World Wide Web*, pages 406–414, Hong Kong, Hong Kong, May 2001.

[34] Jason B Forsyth and Thomas L Martin. Tools for Interdisciplinary Design of Pervasive Computing. *International Journal of Pervasive Computing and Communications*, 8(2):112–132, 2012.

[35] David Franklin and Joshua Flaschbart. All gadget and no representation makes Jack a dull environment. In *Proceedings of AAAI 1998, Spring Symposium on Intelligent Environments*, pages 155–160, Palo Alto, USA, March 1998.

[36] Bob Hardian, Jadwiga Indulska, and Karen Henricksen. Balancing Autonomy and User Control in Context-Aware Systems - a Survey. In *Proceedings of PerCom 2006, Fourth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 51–56, Pisa, Italy, March 2006.

[37] Karen Henricksen. *A Framework for Context-Aware Pervasive Computing Applications*. PhD thesis, University of Queensland, 2003.

[38] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-Awareness on Mobile Devices - the Hydrogen Approach. In *Proceedings of HICSS*

*2003, 36th Annual Hawaii International Conference on System Sciences*, Hawaii, USA, January 2003.

[39] Jan Humble, Andy Crabtree, Terry Hemmings, Karl-Petter Åkesson, Boriana Koleva, Tom Rodden, and Pär Hansson. "Playing with the Bits" User-configuration of Ubiquitous Domestic Environments. In *Proceedings of UbiComp 2003, 5th International Conference on Ubiquitous Computing*, pages 256–263, Seattle, USA, October 2003.

[40] William Jones and Jaime Teevan. *Personal Information Management*. University of Washington Press, 2007.

[41] William P. Jones. *Keeping Found Things Found the Study and Practice of Personal Information Management*. Morgan Kaufmann Publishers, 2008.

[42] David Kirsh. A Few Thoughts on Cognitive Overload. *Intellectica*, 30(1):19–51, 2000.

[43] Andrew J Ko and Brad A Myers. Finding Causes of Program Output with the Java Whyline. In *Proceedings of CHI 2009, ACM Conference on Human Factors in Computing Systems*, pages 1569–1578, Boston, USA, April 2009.

[44] Todd Kulesza, Weng-Keen Wong, Simone Stumpf, Stephen Perona, Rachel White, Margaret M Burnett, Ian Oberst, and Andrew J Ko. Fixing the Program My Computer Learned: Barriers for End Users, Challenges for the Machine. In *Proceedings of IUI 2009, ACM International Conference on Intelligent User Interfaces*, pages 187–196, Sanibel Island, USA, February 2009.

[45] Brian Y Lim and Anind K Dey. Assessing Demand for Intelligibility in Context-Aware Applications. In *Proceedings of UbiComp 2009, 11th International Conference on Ubiquitous Computing*, pages 195–204, Orlando, USA, 2009.

[46] Brian Y Lim and Anind K Dey. Toolkit to Support Intelligibility in Context-Aware Applications. In *Proceedings of UbiComp 2010, 12th ACM International Conference on Ubiquitous Computing*, pages 13–22, Copenhagen, Denmark, September 2010.

[47] Brian Y Lim and Anind K Dey. Design of an Intelligible Mobile Context-Aware Application. In *Proceedings of MobileHCI 2011, 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, pages 157–166, Stockholm, Sweden, August 2011.

[48] Thomas W. Malone. How People Organize Their Desks? Implications for the Design of Office Information Systems. *ACM Transactions on Office Information Systems (TOIS)*, 1(1):99–112, 1983.

[49] Daniel Salber, Anind K Dey, and Gregory D Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of CHI '99, ACM Conference on Human Factors in Computing Systems*, pages 434–441, Pittsburgh, USA, May 1999.

[50] Bill Schilit, Norman Adams, and Roy Want. Context-Aware Computing Applications. In *Proceedings of WMCSA 1994, First Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, USA, December 1994.

[51] Bill N Schilit and Marvin M Theimer. Disseminating Active Map Information to Mobile Hosts. *Network, IEEE*, 8(5):22–32, 1994.

[52] Larry Squire. Memory Systems of the Brain: A Brief History and Current Perspective. *Neurobiology of Learning and Memory*, 82:171–177, 2004.

[53] Thomas Strang and Claudia Linnhoff-Popien. A Context Modeling Survey. In *Proceedings of UbiComp 2004, Sixth International Conference on Ubiquitous Computing, Workshop on Advanced Context Modelling, Reasoning and Management*, Nottingham, UK, September 2004.

[54] Yasuyuki Sumi, Tameyuki Etani, Sidney Fels, Nicolas Simonet, Kaoru Kobayashi, and Kenji Mase. C-MAP: Building a Context-Aware Mobile Assistant for Exhibition Tours. In *Community Computing and Support Systems*, volume 1519 of *Lecture Notes in Computer Science*, pages 137–154. Springer, 1998.

[55] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David Karger. The Perfect Search Engine is Not Enough: A Study of Orienteering Behavior in Directed Search. In *Proceedings of CHI 2004, ACM Conference on Human Factors in Computing Systems*, pages 415–422, Vienna, Austria, April 2004.

[56] Sandra Trullemans. Personal Cross Media Information Management. Master's thesis, Vrije Universiteit Brussel, 2013.

[57] Sandra Trullemans and Beat Signer. From User Needs to Opportunities in Personal Information Management: A Case Study on Organisational Strategies in Cross-Media Information Spaces. In *Proceedings of*

*DL 2014, International Conference on Digital Libraries*, London, UK, September 2014.

[58] Sandra Trullemans and Beat Signer. Towards a Conceptual Framework and Metamodel for Context-Aware Personal Cross-Media Information Management Systems. In *Proceedings of ER 2014, 33rd International Conference on Conceptual Modelling*, Atlanta, USA, October 2014.

[59] Endel T. Tulving and Wayne Donaldson. *Organization of Memory*. Academic Press, 1972.

[60] Hans van der Heijden. Ubiquitous Computing, User Control, and User Performance: Conceptual Model and Preliminary Experimental Design. September 2003.

[61] Geert Vanderhulst, Kris Luyten, and Karin Coninx. ReWiRe: Creating Interactive Pervasive Systems that cope with Changing Environments by Rewiring. In *Proceedings of 2008 IET, 4th International Conference on Intelligent Environments*, pages 1–8, Seattle, USA, July 2008.

[62] Jo Vermeulen. Improving Intelligibility and Control in Ubicomp. In *Proceedings of UbiComp 2010, 12th ACM International Conference on Ubiquitous Computing - Adjunct*, pages 485–488, Copenhagen, Denmark, September 2010.

[63] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems (TOIS)*, 10(1):91–102, 1992.

[64] Andy Ward, Alan Jones, and Andy Hopper. A New Location Technique for the Active Office. *Personal Communications, IEEE*, 4(5):42–47, 1997.

[65] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, 1991.

[66] Steve Whittaker and Candace Sidner. Email Overload: Exploring Personal Information Management of Email. In *Proceedings of CHI 1996, ACM Conference on Human Factors in Computing Systems*, pages 276–283, Vancouver, Canada, April 1996.

[67] Terry Winograd. Architectures for Context. *Human-Computer Interaction*, 16(2):401–419, 2001.

[68] G Michael Youngblood, Diane J Cook, and Lawrence B Holder. A Learning Architecture for Automating the Intelligent Environment. In *Proceedings of IAAI'05, 17th Conference on Innovative Applications of Artificial Intelligence*, volume 3, pages 1576–1581, Pittsburgh, USA, 2005.