A Multi-layered Context Modelling Approach for End Users, Expert Users and Programmers

Sandra Trullemans
WISE Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
strullem@vub.ac.be

Beat Signer
WISE Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
bsigner@vub.ac.be

AVI, June 07–10, 2016, Bari, Italy

Abstract
Context awareness plays an important role in smart environments and embedded interactions. In order to increase user satisfaction and acceptance, context-aware solutions should be controllable by end users. Over the last few years we have therefore witnessed an emerging trend of visual programming tools for context-aware applications based on simple *"if this then that"* rules. Unfortunately, existing solutions do not support the easy reuse of the *"this"* part in other rules. Further, the desired level of control varies among individuals. In order to let users choose the right level of automation and control, we propose a multi-layered context modelling approach distinguishing between end users, expert users and programmers. We report on our ongoing development of the Context Modelling Toolkit (CMT) consisting of the necessary context modelling concepts as well as a rule-based context processing engine. We further discuss an initial design of the graphical user interface for the presented multi-layered context modelling approach.

Author Keywords
Context-aware systems; context modelling toolkit; end user.

ACM Classification Keywords
H.5.m [Information interfaces and presentation (e.g., HCI)]: Miscellaneous

Introduction

It is well-accepted by the community that we have to achieve a better synergy between context-aware systems and their users by providing control and intelligibility [8]. There is no general rule about the necessary level of control since individual users have varying preferences. As pointed out by Hardian [6], there needs to be a balance between the level of control and automation. In order to provide end users some control, visual programming tools such as the Jigsaw Editor [7] and iCAP [4] as well as commercial applications such as IFTTT[1] have been developed for defining simple *"if situation then action"* context rules. Note that in this case a situation consists of one or more events such as the current temperature. Nevertheless, the complexity of context-aware systems is significantly increased in new forms of smart homes. In such a complex environment, advanced modelling support for end users is required since simple context rules cannot cover all desired smart home behaviour [10].

In this paper we explore the opportunities for advanced end user modelling of rule-based context-aware applications. We propose a multi-layered context modelling approach which supports the seamless transition between different levels of expertise. We further enable the reuse of situations defined by end users. Finally, we discuss an implementation of the Context Modelling Toolkit (CMT) enabling the practical use of our multi-layered context modelling approach.

Background

For decades context-aware frameworks have been developed to facilitate the development of context-aware applications. They often include support for context modelling and reasoning. The SOCAM [5] framework is an example of an ontology-based context-aware framework. Unfortunately, ontology-based approaches are not ideal for dynamic context modelling, given that changing the ontology at runtime introduces potential issues with conflicts and ontology integrity. The Java Context Aware Framework (JCAF) [1] takes an object-oriented approach and forwards the reasoning of context rules to the application layer. This has the effect that context rules are spread over various applications which can lead to issues concerning conflict management and inconsistent application behaviour across client applications. In order to overcome this issue, JCOOLS [9] integrates JCAF with the Drools inference engine. Although JCOOLS is very promising, it lacks support for different expertise levels and reusable user-defined situations. Finally, the most well-known and used context-aware framework in the Human-Computer Interaction (HCI) community is the Context Toolkit by Dey et al. [2]. The framework applies a component-based software design where components are represented by so-called widgets. Each widget processes information, from low-level data input to higher-level abstract context information (i.e. detected situations). In order to foster intelligibility and offer control over context rules, Dey and Newberger extended the Context Toolkit with a middleware that includes the concept of situations [3]. Whenever all the widgets of a given situation are active, the situation will invoke an action of the context services listening for it. While some user control is provided via the customisation of parameters, the reuse of situations in the definition of other situations is not possible.

We can conclude that there is a broad spectrum of context modelling tools ranging from context-aware development frameworks to end-user tools. Nevertheless, existing context modelling solutions do not foresee a seamless transition between the different levels of expertise of end users, expert users and programmers and they do not support the reuse of user-defined situations.

[1] https://ifttt.com

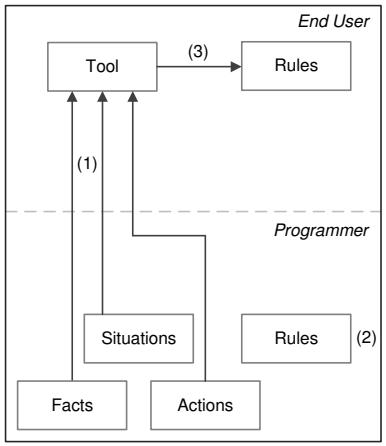

Figure 1: Existing context modelling approaches

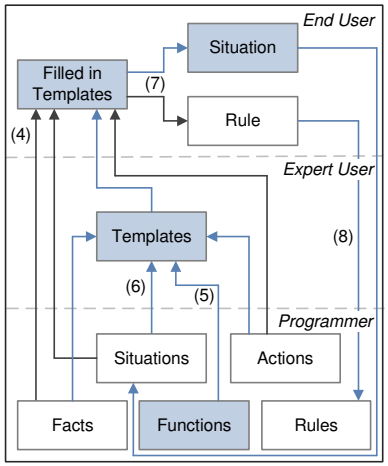

Figure 2: Multi-layered context modelling approach

Multi-layered Context Modelling Approach

In order to facilitate the end-user context modelling process, we foresee the possibility to let end users create new situations such as 'cooking' by creating so-called *situation rules* which take the form of *"if situations then new situation"*. In its simplest form, the IF side of a situation rule can just be a combination of context data as seen in existing visual programming tools, such as iCAP [4], in order to construct context rules. Note that this context data includes facts (real world objects such as Person is Bob), events or situations. Besides simple situation rules, we also support the reuse of newly defined situations on the IF side. For example, the situation rule *"IF Person is Bob and Location is kitchen THEN Bob is in the kitchen"* takes the facts `Person:Bob` and `Location:kitchen` as context data on the IF side to define a new situation. This new situation can then, for example, be reused in the situation rule *"IF Bob is in the kitchen AND stove is on THEN Bob is cooking"* or in a context rule such as *"IF Bob is in the kitchen THEN turn radio on"*. In contrast to existing work, with our approach users do not have to re-define a situation each time they want to use it in a different situation or context rule.

We further make use of *templates* to simplify the definition of situation and context rules. A template serves as a skeleton for rules. The `IF` side can consist of multiple events, situations or logical functions. Similar to programming concepts, a function evaluates some logical statements and returns the result. In our multi-layered context modelling solution, functions only return boolean values. Furthermore, situations and functions can have a number of parameters. For example, a function `IsPersonInLocation` evaluates whether the given `Person` is in the specified `Room` and returns true or false. In contrast to the definition of a context rule, a template does not specify a specific fact or situation for a parameter of a given type. The binding to a specific fact or situation is only done when the template is used to create a concrete situation or context rule. In order to know which object types are needed by the template, it contains a `To Fill In` part listing the required object types which are connected to the corresponding input variables of situations or functions on the `IF` side. When a user fills in a template, they only have to provide the concrete instances of the listed object types such as `Person:Bob`. Finally, the `THEN` side of a template defines the new situation. Since situations can have parameters, template designers can add these to the new situation by defining which properties of a template's required object types have to be forwarded to the new situation. When a template is created, it can be reused to define custom situations or context rules. For instance, the situation rule *"IF Person is Bob and Location is kitchen THEN Bob is in the kitchen"* could be created by using the `PersonIsAtLocation` template by simply filling in the variables `Person:Bob` and `Room:kitchen` and by naming the new situation as *"Bob is in the kitchen"*. The new situation is identified by its name. Similarly, the same template could be used to define that Alice is in the living room.

The situation rules and templates were introduced to support users with different levels of expertise as proposed by Ur [10]. As illustrated in Figure 1, in existing systems facts (e.g. Person is Bob), situations (e.g. cooking) and actions (e.g. turning on the lights) (1) are usually predefined by a programmer and can be applied by an end user to construct simple *"IF situation THEN action"* context rules (3). Furthermore, in context modelling solutions without end-user control, programmers also implement the desired context rules (2). We extended the previous model with an intermediate layer for expert users and integrated the notions of custom and reusable situations as shown in Figure 2. Similar to existing systems, in our multi-layered context modelling approach, programmers are responsible to provide

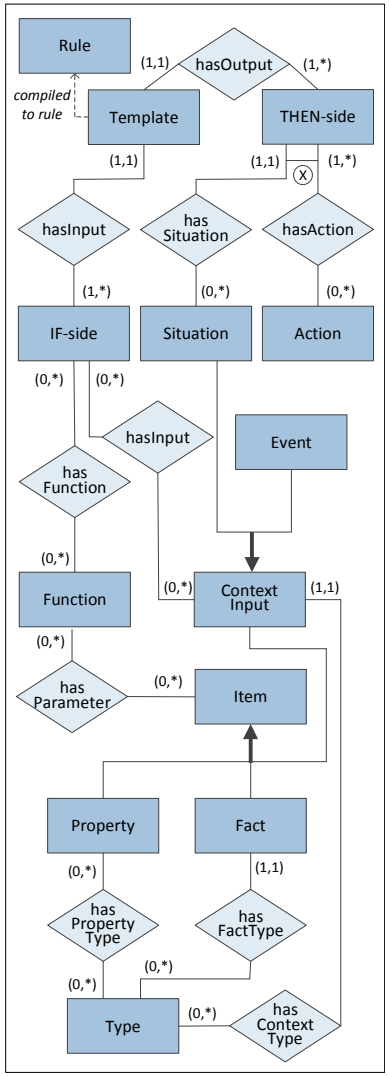

Figure 3: CMT data model

context data in the form of facts, situations and actions to the end user (4). Besides the previously mentioned context data, they also have to provide functions (5) in order that expert users can create templates (6). End users might fill in templates by using the provided context data. In addition to the templates defined by expert users, end users can also use some default templates such as an `AND` template to construct simple situation and context rules. The completed templates define new situation or context rules (7). New situations are made available at the programmer layer in order that they can be reused at the expert and end-user layer (8). Similarly, custom context rules flow from the end-user layer back to the programmer layer and can be used by programmers or other applications. This provides a seamless transition between the three layers.

Context Modelling Toolkit

The proposed multi-layered context modelling approach has been implemented in the Context Modelling Toolkit (CMT). CMT has been designed as a client-server architecture and is implemented in Java. While the server side takes care of the context reasoning, clients such as sensors, third-party applications and applications which allow end users to define new situations, provide the necessary context data to the server. As shown in Figure 4, CMT consists of a db4o[2] database backend and uses the non-persistent version of the Drools 6[3] rule reasoning engine. In order to have a unified representation of the context data delivered by clients, we have designed the conceptual CMT data model shown in Figure 3. The data model highlights the elements of the previously described multi-layered modelling approach in the form of an Entity-Relationship (ER) diagram. The different entities of the CMT data model serve as a common vocabulary used in any client-server communication. We

[2] http://www.mono-project.com/archived/db4o
[3] http://docs.jboss.org/drools/release/6.1.0.Final

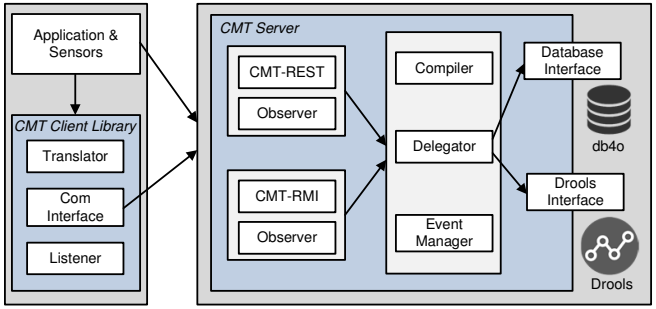

Figure 4: CMT architecture

currently support the Java Remote-Method Invocation (RMI) and REST communication protocols. Since the two protocols require a different data exchange format, the CMT server contains the `CMT-RMI` and `CMT-REST` components. These components translate their input to CMT data model entities and generate the required output format when data is sent to the clients. Finally, the Java-based CMT Client Library consists of a `Translator` component for translating Java objects to CMT data model entities and to the corresponding JSON formats, the `Com Interface` which abstracts REST calls and the `Listener` component which provides Java listeners for CMT.

Clients first have to register the type of situations or facts they will send to CMT. Since Drools requires compiled Java classes of the registered types, the `Compiler` component will compile these classes and add them to the classpath at runtime. After registration, clients can add instances of the registered types to the server. In case the instance is a fact, the `Delegator` component sends it to the `Database Interface` which inserts it into db4o. The `Delegator` component further sends both, facts and situations, to the `Drools Interface` where they are inserted in Drools and

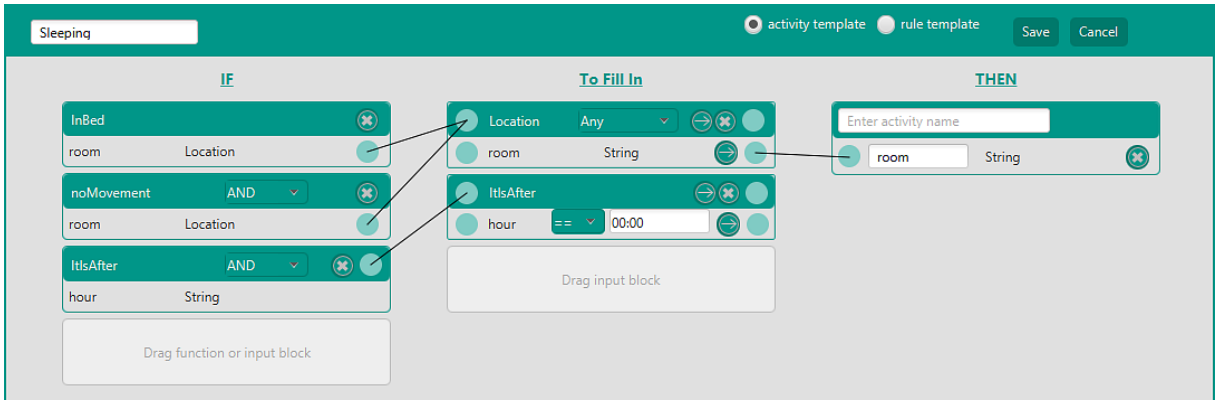

Figure 5: Design of templates in the expert graphical user interface

rules are re-evaluated. Furthermore, new templates are stored in the database. When a filled in template is received, it is compiled to the Drools DRL rule format. The new rule is then inserted into the Drools knowledge base and all rules are re-evaluated. Note that our implementation of the rule compilation is a complex mechanism which takes into account the full first order logic and provides in depth error handling for client applications. When Drools detects some conflicting rules, the `Event Manager` forwards them to the clients. In the future, we plan to further develop this component to provide some intelligibility.

Graphical User Interfaces

End users as well as expert users can interact with CMT via a graphical user interface (GUI). The expert GUI enables the design of templates as shown in Figure 5. In our proof of concept scenario Alice designs a template to create "*Sleeping*" situations. First, she adds to the *IF* side the `inBed` situation which has been defined by a programmer and triggers when a pressure sensor in the bed detects some pressure. Alice also adds the function `noMovement` which again has been defined by a programmer and returns `true` if there is no movement in a given room. Further, she adds a time constraint which indicates that it must be later than a specified time. When the logical statements are defined, Alice has to add the object types to be provided when using the template. In our example, a `Location` and `ItIsAfter` type are added to the *To Fill In* part. Next, she connects the `Location` type to the `inBed` and `noMovement` parameters to indicate that the entered location has to be passed to the two functions. Finally, she allocates the room to the new situation on the *THEN* side by clicking the arrow of the room field in the `Location` type. After saving the template, Alice can use the template to define "*Sleeping*" situations such as that she or her son Bob sleep.

The end user GUI shown in Figure 6 displays the created template with a placeholder for the required `Location` instance where Alice can enter Bob's room. In addition, there is a `ItIsAfter` placeholder. By clicking the 'Enter Values' button, Alice can enter the desired time (e.g. 20:00). She then enters "Bob Sleeps" as the name for the new situation

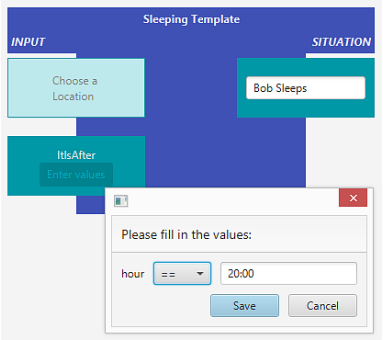

Figure 6: End-user GUI to define new situations

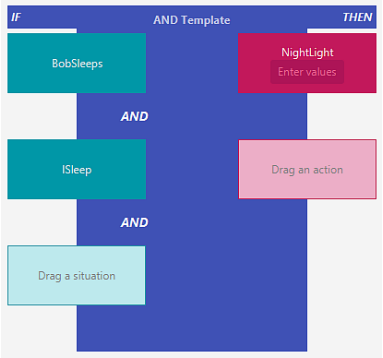

Figure 7: End-user GUI to define new context rules by using the default AND template

on the *Situation* side. Similarly, she can enter her room and specify 22:00 to define that she sleeps. Finally, the sleeping situations can be used to define context rules. Alice can add "Bob Sleeps" and "I Sleep" to the IF side of the AND template (i.e. default context rule template) and add the action that the night light has to be turned on to the THEN side as shown in Figure 7. Note that similar to the creation of templates for new situations, expert users can design templates for advanced context rules.

Conclusion and Future Work

We are currently investigating the further design of the expert user and end user graphical user interfaces. Besides usability and interaction enhancements, we are exploring the integration of feedforward and intelligibility features. Furthermore, we plan to deploy CMT under a research license in order that the HCI community can explore new opportunities in designing user-centric context-aware applications.

We have presented a multi-layered context modelling solution and the corresponding CMT data model as an innovative approach for end users to control their context-aware applications. The presented solution contributes to the existing body of work by enabling end users to define new custom situations in the form of "*if situations then new situation*" at runtime and to reuse these situations in other situation and context rules. Ultimately, the proposed multi-layered context modelling approach enables end users, expert users as well as programmers to collaboratively develop their context models.

Acknowledgements

The research of Sandra Trullemans is funded by the Agency for Innovation by Science and Technology in Flanders (IWT). We would further like to thank Wouter Mensels and Brecht De Rooms for their work on an earlier version of the toolkit.

References

[1] Jakob Eyvind Bardram. 2005. The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. In *Proc. Pervasive*.

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

[3] Anind Dey and Alan Newberger. 2009. Support for Context-Aware Intelligibility and Control. In *Proc. CHI*.

[4] Anind Dey, Timothy Sohn, Sara Streng, and Justin Kodama. 2006. iCAP: Interactive Prototyping of Context-Aware Applications. In *Proc. PerCom*.

[5] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. 2004. An Ontology-based Context Model in Intelligent Environments. In *Proc. CNDS*.

[6] Bob Hardian, Jadwiga Indulska, and Karen Henricksen. 2006. Balancing Autonomy and User Control in Context-Aware Systems - a Survey. In *Proc. PerCom*.

[7] Jan Humble, Andy Crabtree, Terry Hemmings, Karl-Petter Åkesson, Boriana Koleva, Tom Rodden, and Pär Hansson. 2003. "Playing with the Bits" User-configuration of Ubiquitous Domestic Environments. In *Proc. UbiComp*.

[8] Brian Lim and Anind Dey. 2009. Assessing Demand for Intelligibility in Context-Aware Applications. In *Proc. UbiComp*.

[9] Jongmoon Park, Hong-Chang Lee, and Myung-Joon Lee. 2013. JCOOLS: A Toolkit for Generating Context-aware Applications with JCAF and DROOLS. *Journal of Systems Architecture* 59, 9 (2013).

[10] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael Littman. 2014. Practical Trigger-Action Programming in the Smart Home. In *Proc. CHI*.