The Context Modelling Toolkit: A Unified Multi-layered Context Modelling Approach

SANDRA TRULLEMANS, LARS VAN HOLSBEEKE, and BEAT SIGNER, Vrije Universiteit Brussel

Context awareness plays an important role in recent smart environments and embedded interactions. In order to increase user satisfaction and acceptance, these context-aware solutions should be controllable by end users. Over the last few years we have therefore seen an emerging trend towards visual programming tools for context-aware applications based on simple *"IF this THEN that"* rules. However, existing solutions often do not support the simple reuse of the *"this"* part in order to define more sophisticated rules. Given that the desired level of control varies among individuals, we propose a unified multi-layered context modelling approach distinguishing between end users, expert users and programmers. Our Context Modelling Toolkit (CMT) consists of the necessary context modelling concepts and offers a rule-based context processing engine. We further illustrate how end users and expert users might interact with the CMT framework. Finally, we highlight some advantages of our Context Modelling Toolkit by discussing a number of use cases.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing systems and tools**;

Additional Key Words and Phrases: Context-aware systems; context modelling; user control; end-user programming; visual programming; templates.

ACM Reference format:
Sandra Trullemans, Lars Van Holsbeeke, and Beat Signer. 2017. The Context Modelling Toolkit: A Unified Multi-layered Context Modelling Approach. *Proc. ACM Hum.-Comput. Interact.* 1, 1, Article 7 (June 2017), 16 pages.
https://doi.org/10.1145/3095810

1 INTRODUCTION

Fully automated context-aware systems that invoke actions based on sensor input but without taking any user preferences into account have their limitations [2]. Researchers therefore aim to achieve a better synergy between context-aware systems and their users. The findings of Barkhuus and Dey [2] show that users are only willing to change their behaviour and adapt to a fully automated context-aware application if the application brings some major enhancements to their life. Furthermore, fully automated solutions are error prone when developing complex user-centric context-aware applications since users may have different behaviours and routines [3, 17].

As pointed out by Hardian [13], there needs to be a balance between the level of control and automation. For example, the complexity of context-aware systems is significantly increased in new forms of smart homes where the system should adapt to the individual behaviour of different inhabitants [17]. In such a complex environment, advanced modelling support for end users is required since simple "*IF this THEN action*" rules cannot cover all desired smart home behaviour [20].

The research of Sandra Trullemans is funded by the Agency for Innovation by Science and Technology in Flanders (IWT).

2573-0142/2017/6-ART7 $15.00
https://doi.org/10.1145/3095810

The balance between the level of control and automation can be regulated by letting end users personalise context models. However, such a personalisation is far from trivial since users often have diverse conceptualisations of situations. Various applications, such as IFTTT[1], are being developed in order to allow users to define which context cues will trigger certain actions in a context-aware application. Nevertheless, these applications allow users to define only simple context rules in the form of *"IF this THEN that"* or *"IF situation THEN action"*. A situation can, for example, take the form of *"a projector is in use in the meeting room"* and is defined by the application logic. While it is common in automated context-aware systems to abstract situations (i.e. defining a 'meeting' as a concept to reuse it in other situations) [4], visual end-user tools do not offer this functionality. By allowing end users to model situations, more advanced context rules can be defined in a user-friendly manner. Depending on their programming skills, users might define their *"IF situation THEN action"* rules either as expert users or as end users, taking into account that this might influence the expressiveness.

The Context Modelling Toolkit (CMT) presented in this paper contributes to the field of end-user programming for context-aware applications by enabling the seamless transition between different levels of expertise and supporting the definition of new reusable situations.

We start by investigating related work and then provide a general overview of the Context Modelling Toolkit, outlining how new situations can be defined and reused. This is followed by a discussion of our conceptual model for context modelling. Additionally, some implementation details and relevant aspects for developers of CMT clients are provided and completed with the introduction of the end- and expert-user interfaces. Finally, we demonstrate the features and expressiveness of CMT through a number of use cases and conclude with a general discussion and comments about future work.

2 BACKGROUND

Due to the multi-layered approach of our Context Modelling Toolkit which supports programmers as well as end users, we start by presenting context modelling solutions from a programmers perspective and then move to visual end-user context modelling tools.

2.1 Context-aware Frameworks for Smart Homes

For decades context-aware frameworks have been developed to facilitate the development of context-aware applications in various domains such as in smart homes, mobile applications and recommendation systems. Context-aware frameworks often include support for context modelling and reasoning. Various software approaches have been used. The SOCAM [12] framework is an example of an ontology-based context-aware framework. Developers define an ontology including possible situations and their description by using the OWL ontology language. At runtime, the inference engine uses the ontology to reason over low-level contextual data and detects pre-defined situations. Unfortunately, an ontology-based approach is not ideal for dynamic context modelling. Changing the ontology at runtime introduces possible issues with respect to conflicts and ontology integrity.

The Java Context Aware Framework (JCAF) [1] takes an object-oriented approach based on a service-oriented architecture. Context services are components that notify other services or client applications about which situation they have detected and can be added or removed at runtime. By allowing compositions of context services, new situations can be defined. Although this feature is necessary for our proposed approach, JCAF forwards the reasoning of context rules to the application layer. This has the consequence that when users want to change a rule's behaviour, client applications have to be redeployed. Further, context rules are spread over various applications which might lead to issues concerning conflict management and inconsistent application behaviour across client applications. In order to overcome these issues, JCOOLS [18] integrates JCAF with the

[1]https://ifttt.com

Drools inference engine. By defining an application-specific XML schema, client applications can create new facts and events. These are then converted to context services in JCAF and can be used in Drools rules. Although JCOOLS offers a good software basis for our proposed multi-layered context modelling approach, they lack support for end users and expert users.

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. [7]. The framework is based on 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). Client applications can listen to multiple widgets and start reasoning over their context rules to determine which actions should be taken. Similar to JCAF, the context reasoning is pushed to the application layer. In order to allow intelligibility and control over context rules, Dey and Newberger extended the Context Toolkit with middleware that includes the concept of situations [9]. A situation takes input from various widgets. Whenever all of a situation's widgets are active, the situation will invoke an action of the context services listening to it. User control is provided via the customisation of parameters. If users desire to include more restrictions in a situation's logic or change its actions, a new situation has to be compiled at runtime. Finally, the reuse of situations is impossible since situations can only take widgets as input.

2.2 Visual End-User Context Modelling Tools

In the research fields of ubiquitous computing and end-user programming, plenty of research has been done around visual end-user programming tools. Due to our focus on the *"IF situation THEN action"* paradigm, we only include rule-based user interfaces. The Jigsaw Editor [15] is a very simple puzzle-based tool to configure smart homes. Situations and actions are represented by puzzle blocks which users can connect to each other. The puzzle blocks are provided by developers and users cannot change the behaviour of a situation (e.g. what should be triggered to make the situation true). Due to the intentional simplicity of the tool, the available expressiveness is limited. In contrast, iCAP [10] provides a wide range of functionality for context modelling. Users can easily combine context cues to define a situation. For example, person 'Joe' can be combined with location 'Kitchen' to indicate that Joe has to be in the kitchen. These situations are then associated with desired actions. In addition, users can define new facts such as a new person who's name is Bob. The iCAP user interface is built on top of the Context Toolkit which implies that it has the same limitations such as the fact that users cannot define and reuse new situations. Fortunately, due to the loose coupling with the Context Toolkit, iCAP can be integrated on top of other context-aware frameworks. More recent work follows a template approach in order to facilitate the construction of rules. For example, in AppsGate [6] the programmer creates templates for rules where the end user can fill in desired devices and parameters. A similar approach is taken in the work of Ghiani et al. [11] where the presented authoring tool enables programmers, domain experts and end users to define rules for context-dependent cross-device user interfaces.

Over the last few years, commercial applications for defining rules over smart devices are gaining attention. IFTTT is an online service where users can define simple rules via so-called recipes. A recipe consists of a trigger (i.e. situation) and an action. Recipes, triggers and actions can be shared among users. The WigWag[2] mobile application foresees a more expressive platform by allowing users to combine multiple situations with multiple actions. The situations and actions are delivered by the WigWag community. Besides the basic rule-based user interfaces, new approaches to digitally program smart homes or objects have been introduced recently. CAMP [19] steps back from the rule-based approach by allowing users to declare application behaviour in natural language. The tool includes a limited set of magnetic labels with each label defining a concept or action such as 'set'. Although CAMP is a nice programming paradigm, in the context of context modelling it has its limitations.

[2]http://www.wigwag.com/wigwag relay.html

It would be hard to define all possible context elements (situations and actions) beforehand. A more radical approach in end-user context modelling is the introduction of the *programming by example* concept. Systems such as PiP [5], a CAPpella [8] and GALLAG strip [16] allow users to define situations by physically performing the activity. This programming concept is natural with regards to how situations are defined. For user interfaces designed to define rules, it is less suitable since rule modelling requires more cognitive effort [21]. Finally, Reality Editor [14] uses augmented reality to add behaviour to objects. Users can use the mobile phone as a see-through lens in order to, for example, add digital behaviour to a lamp.

Table 1. Comparison of related work based on the type of user who is responsible to define new situations or rules and the potential reuse of situations which is either not possible (-) or possible but not forseen (+/-)

Application	**Situations defined by**	**Rules defined by**	**Reusable situations**
SOCAM [12]	programmer	programmer	-
ICAF [1]	programmer	programmer	-
JCOOLS [18]	programmer	programmer	+/-
Context Toolkit [9]	programmer and customisation by user	programmer	-
Jigsaw [15]	programmer	user	-
ICAP [10]	programmer and customisation by user	user	-
CAMP [19]	programmer	user	-
AppsGate [6]	programmer and customisation by user	user	-
Ghiani et al. [11]	programmer and customisation by user	programmer, domain expert and user	-

Based on our investigation of related work, we can conclude that there is a broad spectrum of context modelling tools ranging from context-aware frameworks for developers to end-user tools. Recently, Ur et al. [20] aimed for the development of tools that provide some distinct degree of complexity since end users are very different in their abilities to define *"IF situation THEN action"* rules. However, existing context modelling tools commonly allow users to customise parameters of situations which are pre-defined by programmers and rules are mostly defined by the programmer or the user as shown in Table 1. We can also observe that the work of Ghiani et al. [11] identified the need for three types of users in the design of context rules as proposed by Ur et al. [20]. Nevertheless, similar as to the majority of related context modelling tools, they do not foresee the possibility that users can define and reuse custom situations. In contrast, our Context Modelling Toolkit enables programmers, expert users as well as end users to create situations and rules at their level of expertise and allows all three types of users to reuse each others' situations in order to simplify the definition of the IF side of a rule.

3 CONTEXT MODELLING TOOLKIT

The Context Modelling Toolkit (CMT) is a context modelling framework which enables a seamless transition between the different levels of expertise of *programmers* who are experienced with low-level programming, *expert users* with little or no programming experience as well as *end users* without any programming knowledge.

Programmers are responsible for providing interfaces with devices and applications since this step requires programming skills. Expert users can create templates for rules which only requires some basic knowledge about data types and declarative concepts. Finally, end users can fill in rule templates with data instances such as a specific device but do not have to be familiar with the internal logic of a rule. A user can switch between the three user types if they have the necessary expertise for a particular level. The CMT framework further enables the creation of new situations at runtime and fosters the reuse of the custom situations at all three user levels.

The CMT framework is implemented as a rule-based client-server architecture. While the server side takes care of the context reasoning, clients feed the server with context data such as facts (i.e. persistent data such as a person), situations (i.e. temporal data such as the current temperature) and actions which can be invoked when situations are detected. Additionally, custom-defined situations from end users are also sent to the server by the graphical end-user context modelling user interface. Based on this context data, programmers, expert users and end users can create context rules which will be evaluated by the server every time new sensor events are received. When context rules become valid, the assigned actions are invoked.

In order to facilitate the end-user context modelling process, we foresee the option 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 events as seen in existing visual programming tools, such as ICAP [10], in order to construct context rules. Nevertheless, we also allow users to reuse 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 `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 on radio"*. In contrast to previous work, with our approach users do not have to re-define a situation every time they want to use it in a different situation or context rule.

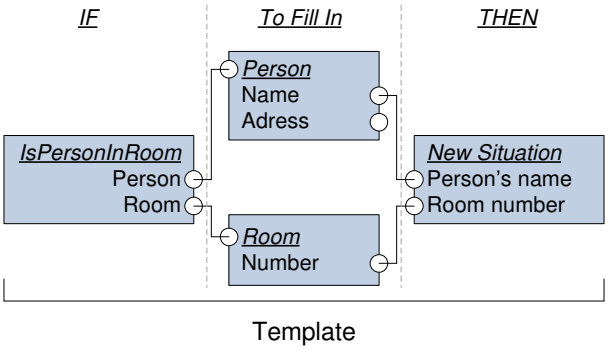
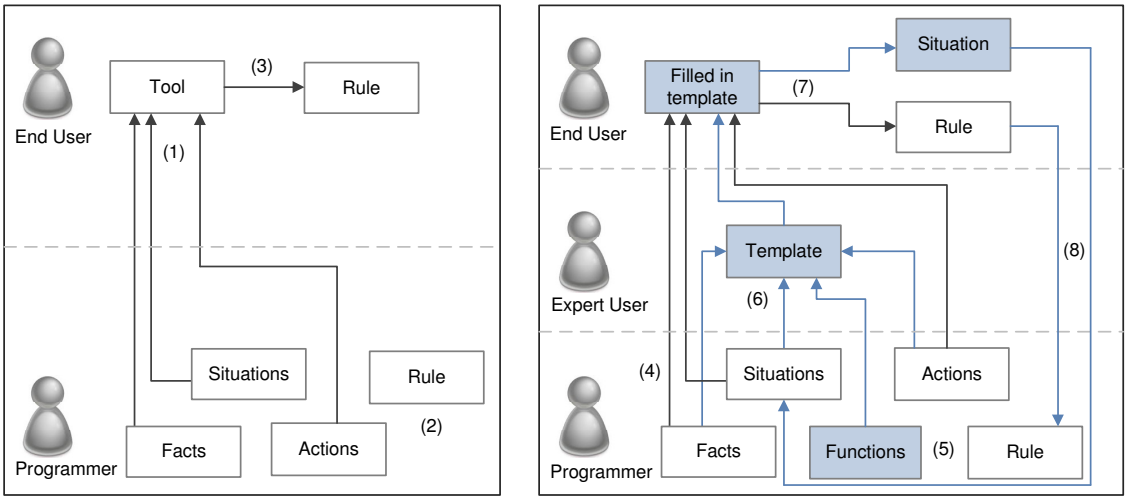


Fig. 1. `personIsAtLocation` template

We use the concept of *templates* to further simplify the definition of situation and context rules. A template serves as a skeleton for rules as illustrated in Figure 1. The IF side can consist of multiple situations or logical functions. Similar to programming concepts, a function evaluates some logical statements and returns the result. In a CMT context, functions can only return boolean values. For example, a function might evaluate whether two persons are at the same location. Situations and functions might take various parameters such as a person and a room which have a specified type (e.g. `Person` and `Location`). In contrast to the definition of a rule, a template does not specify a specific instance (e.g. `Person:Bob`) for a parameter of a given type. The binding to a specific instance is only done when the template is used to create a concrete situation or context rule. In order to



(a) Existing context modelling approaches

(b) Multi-layered CMT approach

Fig. 2. Differences between existing context modelling approaches and our multi-layered CMT solution

specify the required object types, the template contains a `To Fill In` part listing the required object types. The listed object types 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, expert users can add these to the new situation by defining which properties of the template's required object types have to be forwarded to the new situation. For example, in Figure 1 the person's name and the room number are forwarded to the new situation. After a template has been created, it can be used to define situation and context rules. For example, 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 shown in Figure 1 and by simply filling in the variables `Person:Bob` and `Location:kitchen` followed by naming the new situation as *"Bob is in the kitchen"*. The same template could be used to define that Bob is in the living room.

3.1 Multi-layered Context Modelling

The situation rules and templates were introduced to support users with different levels of expertise as proposed by Ur et al. [20]. In existing systems, programmers are often responsible for providing context data in terms of facts and situations as well as possible actions (1) which an end user can then apply to construct simple context rules (3) as illustrated in Figure 2a. Facts such as `Person:Bob` and actions like turning on the lights are usually defined once by the programmer. In contrast, sensors or third-party applications are programmed to send situation events such as the current temperature in a stream of data. Furthermore, in context modelling solutions without end-user control, programmers also implement the desired context rules (2). In contrast to existing work, we extended the previous model with an intermediate layer for expert users and integrated the notion of custom and reusable situations. In addition, custom created rules and situations flow back to the programmer layer to provide a seamless transition between the three layers as shown in Figure 2b. Each layer is designed for users with a specific profile as previously mentioned.

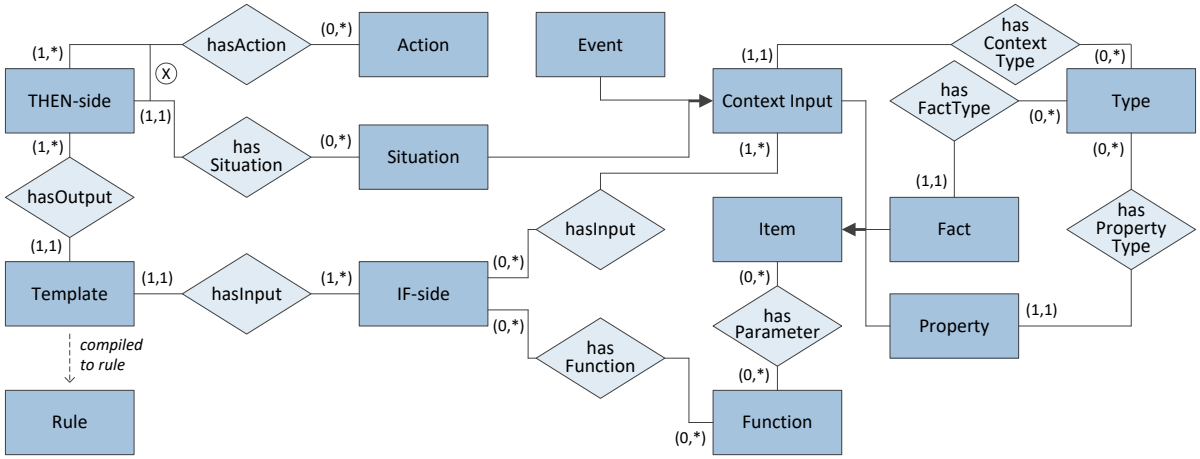


Fig. 3. CMT data model

Similar to existing systems, in our multi-layered context modelling approach shown in Figure 2b programmers are responsible to provide facts, situation events and actions to the end user (4). In order to allow expert users to create templates, they also have to provide functions (5) besides the previously mentioned context data to the expert user (6). End users can fill in templates by using the provided context data. The completed templates can define new situation and context rules (7). New situations are made available at the programmer layer in order that they can be reused on the expert and end-user layer (8). Similarly, custom context rules flow from the end-user layer back to the programmer layer. These custom rules are then usable by programmers or other applications to reason over.

Our approach does not only provide end users control in terms of context modelling but also with respect to the balance between automation and context modelling control. For example, some users might only design new context rules or situations when they have a real need for specific functionality while leaving simple context modelling to a programmer. Other users might design their entire context-aware environment, such as their smart home, themselves.

4 IMPLEMENTATION

In order to have a unified representation of the context data sent by sensors, applications or end users, we have designed the CMT data model shown in Figure 3. The CMT data model simply models the previously discussed templates, functions and context data. Items such as facts and context input require a certain object type (e.g. `Person`) which can have multiple properties (e.g. a person's name). Context input can either consist of events or situations. While events are context input object types pre-defined by a programmer, situations are defined via a situation rule by the end user or programmer. As elaborated earlier, templates have an IF side and a THEN side. The IF side might include context input and functions. In turn, functions can have context input or facts as parameters. In addition, these parameters can refer to one of the properties of a context input or fact object type. Furthermore, a template can be used to construct situation or context rules. In the case of a situation rule, the THEN side of the template must include a new situation. On the other hand, in a context rule actions can be added to the THEN side.

The Context Modelling Toolkit is designed as a client-server architecture implemented in Java. As shown in Figure 4, CMT has a db4o[3] database backend and uses Drools 6[4] as rule engine. We currently support the Java Remote Method Invocation (RMI) and REST communication protocols. Since the two protocols require a different data exchange format (i.e. RMI shares classpaths and REST uses JSON), 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 clients.

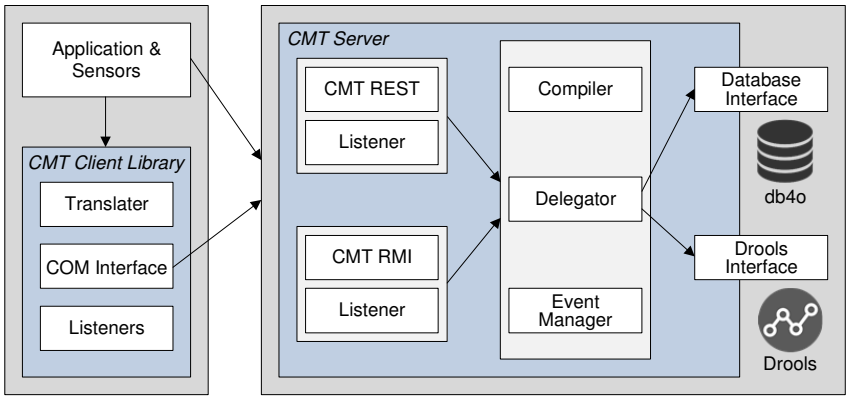


Fig. 4. CMT architecture

In order that clients can reuse each others context model and the server can reason over the clients' context models, clients have to follow the steps highlighted in Table 2. First, a client has to register the object types of facts, situations and actions which they use in their context model. Depending on the used communication protocol, the CMT-RMI or CMT-REST component will translate the incoming data to `Fact`, `Event` or `Action` objects which are defined by the CMT data model. Next, the `Compiler` component will check if the object type already exists based on the given type name. In case it is, the client is notified with an error. Since Drools requires compiled Java classes of the registered object types, the Compiler component will compile these classes and add them to the classpath at runtime. The server then makes these object types available to all other clients which can reuse these context elements in available functions, templates and rules.

Client		Server
1) **register** object types of facts, events and actions	→	make types available
2) **add** facts, situations, functions and templates	→	make objects available and evaluate rules
3) **fill in** template	→	compile and evaluate rules
4) **listen**	←	notify about state changes

Table 2. Steps performed by a CMT client

[3]https://sourceforge.net/projects/db4o/
[4]https://www.drools.org

After registration, clients can add instances of the registered object types to the server. In case of the REST version, a new class instance will be created and the appropriate field values will be set to the values given in the incoming JSON format. The CMT-REST and CMT-RMI components will forward the received fact or event object to the `Delegator` component shown in Figure 4. In case the object is a fact, the `Delegator` component sends it to the `Database Interface` which inserts it into db4o. The `Delegator` further sends both, facts and events, to the `Drools Interface` where they are inserted in Drools and rules are re-evaluated.

When using the REST interface to add templates and functions to the CMT server, the JSON format is again translated to `Template` and `Function` objects which are compiled and made available to Drools as well as to the clients. Furthermore, templates are stored in the database. When a filled in template is received, it is compiled to Drools' DRL rule format. This 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.

Clients can also listen to server state changes. State changes include changes in the data model such as added facts or situations but also actions which are triggered or situations which became valid. The CMT-RMI and CMT-REST `Listener` components are in charge of providing the appropriate mechanism to notify their clients. For example, the REST version implements a publish-subscribe pattern where clients can subscribe to various event types via a WebSocket connection. When using RMI, clients can implement the `CMTEventListener` interface which includes event notifications of, for example, new situations that are added to the CMT server.

Finally, there needs to be a conflict management mechanism for actions associated with situations. Moreover, in the case that actions are associated with a situation in a context rule and the same situation is used in a situation rule, the actions will be invoked when the situation becomes valid—in addition the actions related to the situation defined by the situation rule are also invoked. For example, if the situation 'meeting' is re-used in a 'department meeting' situation rule and has been used in a context rule with actions, it might be desired to only invoke actions linked to the department meeting and not those of the more generic meeting situation. To handle such a use case, the `Event Manager` component is introduced. When situations become valid, the `Event Manager` is notified. It catches the triggered action events from Drools. Before notifying clients to invoke these actions, the Event Manager checks whether there is no other valid situation triggered by the previous situation. When this recursive process stops, clients are notified with the last valid situation and its corresponding actions.

5 CONTEXT MODELLING TOOLKIT USAGE

The multi-layered context modelling approach requires different forms of interaction with the CMT server. While programmers can directly use the CMT REST interface, expert users and end users need a graphical user interface.

5.1 Programmer

We currently provide a CMT Client library in Java for the easy development of Java and Android applications making use of CMT. As known in programming, applications have their own data representation. For example, a Java application can have the class `Person` to represent people. When a client wants to add an application-specific type to the CMT server (e.g. a class `Person` in Java) in order that end users can use this type of context data in their context modelling, it has to be translated to a `Fact` instance of our CMT data model. To facilitate this conversion, methods are provided by the CMT Client library. For example, a programmer just has to provide the class instance and the field name which represents the unique identifier of a fact when registering the fact type as shown on line 1 in Listing 1.

Listing 1. Registering and adding a fact by using the CMT Client library

```
1 registerFacttypeInCMT (Person.class,"name");
2 addFactInCMT(personBob);
```

In order to send a fact instance to the CMT server, the appropriate method can be called and the Java object representing the fact (e.g. Person instance with name Bob as unique identifier) can simply be given as parameter as shown on line 2 in Listing 1. It is the responsibility of the `Transformer` component included in the CMT Client library to do the necessary conversions to the CMT data model whereas the `COM Interface` provides abstractions for the REST calls to the CMT server.

Listing 2. Registering and adding a fact by using the REST interface

```
1 {"className":<Person>, "uriField": <name>, "fields":[{"fieldName":<name>,
      "type":<String>},{"fieldName":<age>, "type":<Integer>}]}
2 {"className":<Person>, "uriField": <Bob>, "fields":[{"fieldName":<age>, "fieldValue":"55"}]}
```

Finally, Java applications can implement listeners, available in the `Listeners` component, which catch specific events sent by the server. Note that it is not mandatory to use the CMT Client library. Clients can also communicate with the CMT server via REST calls as illustrated in Listing 2.

5.2 End-User and Expert-User Interface

In order to support end-user context modelling and allow expert users to create templates, we propose two initial versions of our graphical user interfaces. Note that in the near future we plan to further enhance these GUIs with more advanced interaction techniques such as *programming by example* and validate the proposed user interfaces.

Both graphical user interfaces are implemented in one JavaFX application which uses the CMT Client library in combination with the REST CMT server version. Due to our aim of a seamless transition between expertise levels, the application has an end-user and an expert-user mode. Users can freely switch between the two modes based on their skills or needs. The application consist of a desktop space in the centre surrounded by context data boxes as shown in Figure 5. The left-hand side of the general graphical user interface view contains the context data boxes with time-based events (1), persons (2), locations (3), objects (4), custom situations (5) and situations provided by developers (6). We distinguish between time-based events, persons, locations and other objects because users often think in these distinctive dimensions as mentioned by Dey [10]. The context data elements in these boxes can be mapped back to CMT data model instances of `Fact` and `Context Input`. Furthermore, the right-hand side contains the available templates (7), actions (8) and the defined context rules (9). When the user works in expert mode, a box with the available functions is added to the right-hand side.

In order to define new context rules in end-user mode, end users can use the 'AND template' which takes situations or time-based events as input on the IF side as shown in Figure 5. In this template, the logical operator is limited to AND operations in order to keep it simple. However, it is up to the programmer or expert user to develop more advanced templates to, for example, support more complex logical statements. The THEN side can be populated with actions with the corresponding properties. In the example shown in Figure 5, user Sandra models a context rule stating that *"IF Vince, her child, is sleeping AND she is sleeping THEN the small night light has to be turned on"*. This rule makes sense for her personal household since Vince is afraid of the dark and requires a night light to sleep. However, to save energy, she only wants to turn it on when she is also sleeping. In order to create this context rule, she previously defined the situations which track if Vince or herself are sleeping. She can now also reuse the *"Vince Sleeps"* situation in another context rule, for example, *"IF Vince Sleeps THEN set the TV volume to 20"*.

End users use templates to define a new situation. Expert users can create these templates by switching to the expert mode where the template authoring view is shown in the desktop space as given in Figure 6. Experts can drag functions or events (i.e. situations or time-based events) to the IF side (1). They can choose logical operators such as AND, OR, NOT between added events and functions (2). As defined by the CMT data model, context input and functions may require certain parameters. These parameters are listed below the header which

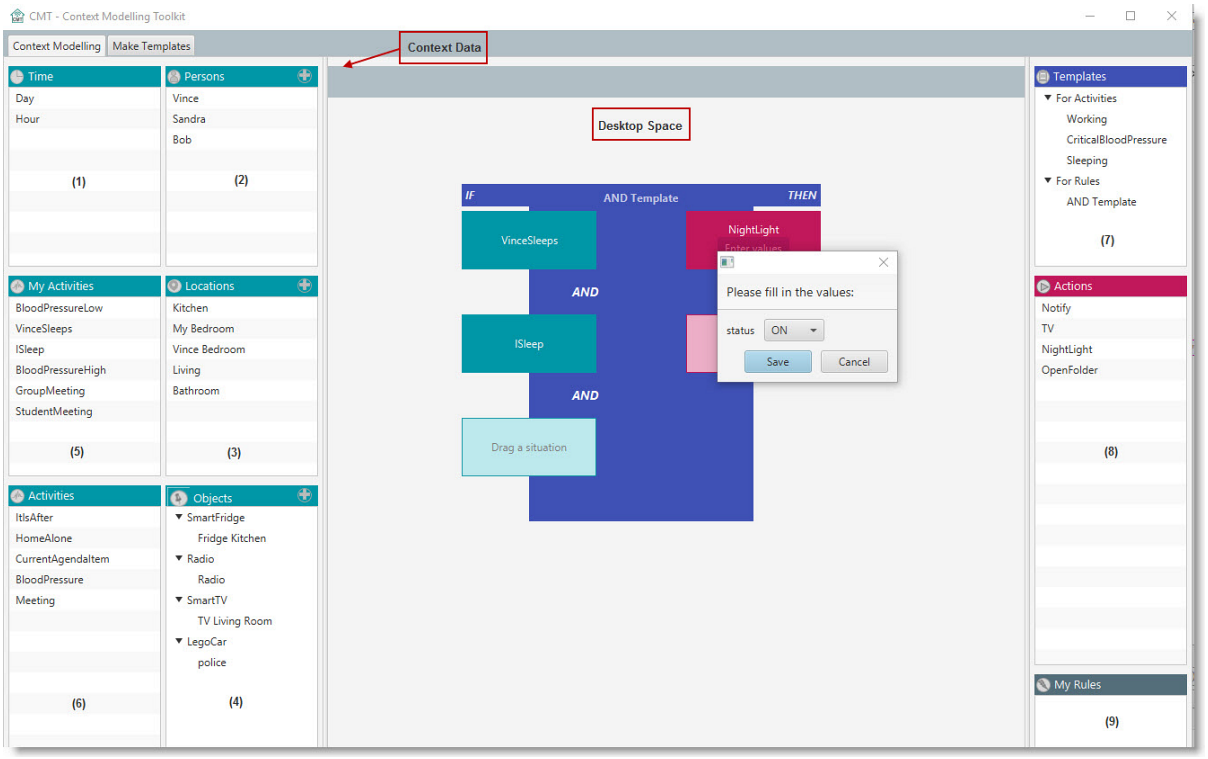


Fig. 5. Main view

includes the function's name or event type (3). Since the template does not define the value of these parameters but only the required object type, the template view includes a "To Fill In" space (4). In this section, the required object types needed to complete the template are listed. Expert users can drag and drop context data object types from the context data boxes provided on the left-hand side of the main view to the empty placeholder (5). In order to keep a reference between these object types and the event or function parameters, links can be created between a parameter and the associated object type (6). Finally, the new situation defined on the THEN side (7) can be given properties by selecting them from the listed object types which have to be filled in (8). After a template has been defined, it is sent to the CMT server and made available to all other client applications. In our example, the template *"Sleeping"* is created as shown in Figure 6. The expert user Sandra defined sleeping as being in bed without any movement in the bedroom and being later than a specified time. In this example, being in bed is a predefined situation provided by a programmer based on a motion sensor that has been placed under the pillow and sends *"in bed"* events. The *"in bed"* event has a bedroom as parameter. The *"no movement"* event also takes a room as parameter and is fired when the motion detector is in idle state. Finally, the *"later than time"* event is fired when the current time is later than the specified time. Our expert user drags a `Location` object type to the "To Fill In" list and connects the `Location` type to the corresponding parameters of the *"in bed"* and *"no movement"* events. Finally, the user leaves the `Location` to state `ANY` (9) without filling in a specific room and does not specify a time for the *"later than time"* event. These values have to be entered by the end user when applying the template. In the case that the expert user defines the value of some of these object types (e.g. by selecting a room instead of `ANY`), the value cannot be changed later by the end user when using the

template. Note that also programmers can provide such templates to end users. This can be desired in the context of application-specific logic such as in a fitness or health application.

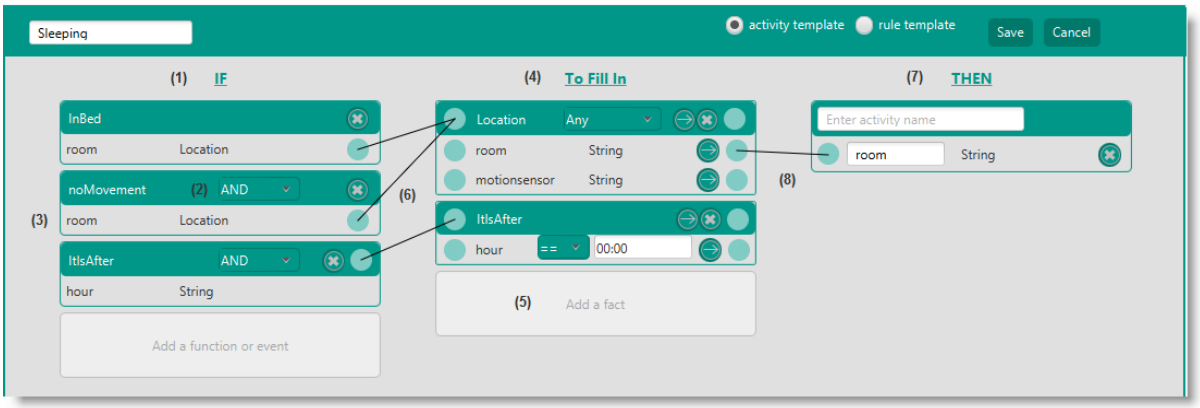


Fig. 6. Template authoring view

End users can use the templates to create new situations by dragging a template to the desktop space where the template's end-user view is shown as illustrated in Figure 7. The template's IF side contains the placeholders for the required object types defined by the expert user in the "To Fill In" list. When the end user selects a placeholder, they are given a request to select an instance of the required object type or to enter necessary data as shown in Figure 7. Finally, when hovering over the template, the logic defined by the expert user will be shown in textual form. In terms of our example, the first placeholder requires an instance of a `Location` while the user only has to enter the desired time in the second placeholder. Now that Sandra has created the *"Sleeping"* template in the expert mode, she can populate it to define the situation where Vince sleeps. She selects *"Vince Bedroom"* as an instance of `Location` for the first placeholder and sets the time value to 8PM. Finally, she labels this new situation as *"Vince Sleeps"*.

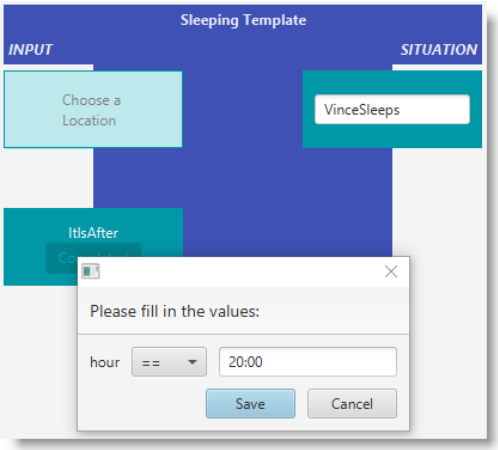


Fig. 7. End User view of a template

As mentioned before, the new situations can be reused in other situation or context rules. Note that without reusable situations or templates, the user would have to redefine the logic of *"Vince Sleeps"* every time they would like to use it in new context rules. In addition, our end user can reuse the template to define the situation where they are sleeping by having their bedroom as a location and a later time constraint.

6 USE CASES

In the following we provide two implemented use cases illustrating the power of the Context Modelling Toolkit.

6.1 Monitoring Elderly People

In the current ageing society, technology-enhanced elderly support becomes more important. Various tools, such as fall detection systems or collaborative health platforms, are developed to help healthcare workers in providing the best care. We show how the use of our CMT can help to further push the development of such healthcare monitoring tools. For example, custom context rules can be defined by healthcare workers based on the available sensors and specific critical situations. Often critical situations vary across patients, which makes it difficult to develop a solution that fits everybody. Let us assume that a patient or their relatives defined the *"ISleep"* situation by using the *"Sleeping"* template as introduced earlier. This situation can then, for example, be used in the context rules shown in Listing 3.

Listing 3. Rule examples using the *"ISleep"* situation

```
1 IF stove == on AND ISleep THEN Notify(central, "alarm")
2 IF BloodPressureLow AND NOT ISleep THEN Notify(nurse, "blood pressure low")
3 IF BloodPressureHigh AND AloneHome THEN Notify(doctor, "critical blood pressure")
```

Note that these rules are highly customised to an individual subject since other patients might require other actions. For example, when they have a high blood pressure, a reminder to take their medication might have to be given to them. In addition, the critical value that determines when the patient's blood pressure is low/high might differ for certain individuals. When using our CMT, healthcare workers can use a template which implements the logic of critical blood pressure measurements where they just have to fill in the desired critical values. Furthermore, when a healthcare worker becomes knowledgeable about CMT, they might switch to the expert mode to define a new template for measuring blood pressure with extra functionality where the heart rate value is also considered. The following building blocks are implemented:

- **Stove is ON**: situation provided by a programmer of the stove sensor. The stove sends events including its ID and state.
- **AloneHome**: fact inserted by a door application in CMT when the healthcare worker leaves the house.
- **BloodPressureValue**: situation sent by the blood pressure device including the current value. Note that this event can also be sent by other blood pressure devices as long as they use the same event object type.
- **BloodPressureLow/High**: situations defined in the end user mode by the healthcare worker by using the *"CriticalBloodPressure"* template with patient-specific critical values.
- **CriticalBloodPressure template**: template designed by an expert user as having a blood pressure measure event with the pressure value as parameter as well as a function which checks whether the current blood pressure is above a certain threshold.

6.2 From Smart Homes to Digital Homes

Not only sensors can provide context data. Environments where digital applications can also form part of the smart ecosystem are so-called Digital Homes. Due to the generic implementation of the CMT framework, any application can add facts, situations, templates or actions to the server. Therefore, the CMT framework can be

used as a customisation tool for implicit digital behaviour. For example, a programmer provided Sandra with a small application that sends *"LaptopActivity"* situation events to the CMT server every time there are mouse or keyboards events detected on her laptop. Sandra used this situation in forming a context rule that says that *"IF LaptopActivity THEN set door display to state busy"*. After a while she gets familiar with CMT and decides to design more custom complex rules by creating her own template. With the previous context rule, she had the issue that the display would also be set on state busy when she is just checking emails. Nevertheless, she still wants to have the display's busy state when, for example, writing a research paper but desires the available state when she edits slides of a course. A programmer has created an extension to the Windows File Explorer which monitors which file is accessed and sends an *"WorkingInFolder"* event including the file and the current directory's name. Sandra can use this event in her template design for generalising a working situation since her papers are grouped in a publications folder and course's slides in the course folder. She defines that the template has to check if the *"WorkingInFolder"* event's current directory is the same as the filled in folder. This template is used to define the situation where she is writing a research paper by filling in the "Publications" folder in the end user view of the previously defined template. She can now also reuse this template in the future to define that she works on a course by filling in the "Course" folder. Finally, she only has to use her new defined situation of writing a research paper in a context rule which sets the door display to busy state when writing, for example, the EICS paper is recognised. Similarly, she can use her working on a course situation in a context rule to set the display state to available. In addition, both custom situations can be reused in the definition of new situations such as *"IF WrittingResearchPaper AND my location is meeting room THEN MeetingAboutPaper"*. This new situation can then be used in a context rule to set her smartphone to silent mode.

We have illustrated that a single user can easily switch between end- and expert-user modes depending on their own developed skills. Tools such as IFTTT have already proven that such an implicit digital behaviour has its advantages for millions of users. Nevertheless, with the provided reuse of self-defined situations and seamless transition between expertise levels, users can program much more functionality. The following building blocks are implemented by a programmer and defined by the end or expert user:

- **DesktopActivity**: situation send by a third-party application developed by a programmer. This event is sent to the CMT server when mouse or keyboard activity is detected.
- **DoorDisplay**: display listens to the CMT server for *"DoorDisplay"* action invocations with a state as property. States can be busy, available or away.
- **WorkingInFolder**: situations defined by a programmer and is sent when the Windows File Explorer extension detects access to a file. The sent situation event includes properties such as the file's name and its directory.
- **Working template**: template designed by Sandra in the role of an expert user. The IF side of the template includes the *"WorkingInFolder"* situation event and the "To Fill In" part contains the `Folder` object type provided earlier by a programmer. The `Folder` object type's folder name property is connected to the folder name property of the *"WorkingInFolder"* event located in the IF side.
- **WritingReseachPaper**: custom situation defined by Sandra by using the *"Working"* template in the end-user mode. The template is filled with the `Folder` instance that represents the "Publications" folder.
- **EditingCourse**: custom situation defined by Sandra. Similar to the above situation, she uses the *"Working"* template in the end-user mode, filled with the "Course" folder.

7 DISCUSSION AND FUTURE WORK

The presented Context Modelling Toolkit supports the seamless transition between end users, expert users as well as programmers. It allows the creation of new situations across all three levels of expertise and makes situations reusable in situation and context rules. In order to achieve this functionality, we have introduced

the concept of situation rules in the form of "*IF situations THEN new situation*" and templates. The approach of defining custom new situations and combining them to more complex situations as well as reusing them is well used in other context-aware frameworks which are based on ontologies or probability-based activity recognition. Nevertheless, these frameworks miss the opportunity to let the user define new situations at runtime. Similarly, rule-based context-aware frameworks such as JCOOLS [18] do not offer the possibility to define composed situations at runtime. While the runtime-compilation of a new situation is possible in Dey's Context Toolkit [7], it would require significant error handling since the context rule is encapsulated in the Context Toolkit's situation [9]. However, situations in the Context Toolkit can only have widgets (i.e. sensor data streams) as input and developers cannot reuse situations to compose more complex situations. While there exists some research on visual rule-based programming tools [10, 15], existing solutions do not foresee a distinction between end users and expert users and they do also not provide any mechanism for letting users at all three expertise levels work in synergy. Our proposed Context Modelling Toolkit together with its multi-layered conceptual model allows end users to have more advanced control over their smart environment and at the same time offers a balance between automation and manual context modelling.

We are currently deploying a longitudinal exploratory user study which includes the validation of the multi-layered context modelling approach as well as the evaluation of the proposed graphical user interfaces for end users and expert users. We will recruit participants for each of the three expertise levels. Each group of participants will first model a context-aware environment with predefined controlled building blocks. In order to gain insights about the seamless transition between the different levels of expertise, participants will additionally model a context-aware environment by using the building blocks designed by participants working on another level of expertise. Furthermore, end users and expert users will model an environment at different retention intervals. In this way, we will be able to observe whether end users might become expert users over time and if expert users can likewise play the role of an end user. Finally, we will observe to which degree intelligibility plays a role in and across the different levels of expertise.

In the future, we also plan to extend the CMT server with some rule conflict management. Although the Drools rule engine already provides some logical conflict resolution, conflicts such as semantically contradicting compositions of situations and nested action triggers still have to be investigated. In addition, intelligibility has to be reviewed due to the possible complex nested definitions of new custom situations. A main question is how we can indicate to a user when a situation is not detected without having to show them the entire logic of the situation's rule? Furthermore, a social platform to share situation/context rules and templates might lead to collaborative context modelling which would be interesting to investigate on social and behavioural aspects such as copying friends' routines to improve their smart home experience.

8 CONCLUSION

We have presented the Context Modelling Toolkit which offers an innovative solution that enables end users to control their context-aware applications. The Context Modelling Toolkit contributes to the existing body of work by providing programmers, expert users and end users the possibility 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. Based on the proposed unified multi-layered context modelling approach, end users, expert users as well as programmers can collaboratively construct their context models. In this way, the end user gets more control over the context modelling phase and can better balance between the desired level of automation and control.

REFERENCES

[1] Jakob E. Bardram. 2005. The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications. In *Proceedings of Third International Conference on Pervasive Computing (PERVASIVE 2005)*. 98–115. DOI: http://dx.doi.org/10.1007/11428572_7

[2] Louise Barkhuus and Anind Dey. 2003. Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Examined. In *Proceedings of the 5th International Conference on Ubiquitous Computing (UbiComp 2003)*. 149–156. DOI: http://dx.doi.org/10.1007/978-3-540-39653-6_12

[3] Victoria Bellotti and Keith Edwards. 2001. Intelligibility and Accountability: Human Considerations in Context-Aware Systems. *Human-Computer Interaction* 16, 2-4 (2001), 193–212. DOI: http://dx.doi.org/10.1207/S15327051HCI16234_05

[4] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. 2010. A Survey of Context Modelling and Reasoning Techniques. *Pervasive and Mobile Computing* 6, 2 (2010), 161–180. DOI: http://dx.doi.org/10.1016/j.pmcj.2009.06.002

[5] Jeannette Chin, Vic Callaghan, and Graham Clarke. 2006. An End-User Programming Paradigm for Pervasive Computing Applications. In *Proceedings of the ACS/IEEE International Conference on Pervasive Services (ICPS 2006)*. 325–328. DOI: http://dx.doi.org/10.1109/PERSER.2006.1652254

[6] Joëlle Coutaz and James Crowley. 2016. A First-Person Experience with End-User Development for Smart Homes. *IEEE Pervasive Computing* 15, 2 (2016), 26–39. DOI: http://dx.doi.org/10.1109/MPRV.2016.24

[7] Anind Dey, Gregory D. 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). DOI: http://dx.doi.org/10.1207/S15327051HCI16234_02

[8] Anind Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. 2004. a CAPpella: Programming by Demostration of Context-Aware Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2004)*. DOI: http://dx.doi.org/10.1145/985692.985697

[9] Anind Dey and Alan Newberger. 2009. Support for Context-Aware Intelligibility and Control. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2009)*. 859–868. DOI: http://dx.doi.org/10.1145/1518701.1518832

[10] Anind Dey, Timothy Sohn, Sara Streng, and Justin Kodama. 2006. iCAP: Interactive Prototyping of Context-Aware Applications. In *Proceedings of the 4th International Conference on Pervasive Computing (PERVASIVE 2006)*. 254–271. DOI: http://dx.doi.org/10.1007/11748625_16

[11] Giuseppe Ghiani, Marco Manca, and Fabio Paternò. 2015. Authoring Context-dependent Cross-device User Interfaces Based on Trigger/Action Rules. In *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia (MUM 2015)*. 313–322. DOI: http://dx.doi.org/10.1145/2836041.2836073

[12] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. 2004. An Ontology-based Context Model in Intelligent Environments. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2004)*. 270–275. http://www-public.tem-tsp.eu/~zhang_da/pub/Ontology-2004-2.pdf

[13] Bob Hardian, Jadwiga Indulska, and Karen Henricksen. 2006. Balancing Autonomy and User Control in Context-Aware Systems – A Survey. In *Proceedings of the 4th International Conference on Pervasive Computing (PERVASIVE 2006)*. DOI: http://dx.doi.org/10.1109/PERCOMW.2006.26

[14] Valentin Heun, James Hobin, and Pattie Maes. 2013. Reality Editor: Programming Smarter Objects. In *Proceedings of ACM Conference on Pervasive and Ubiquitous Computing (UbiComp 2013)*. 307–310. DOI: http://dx.doi.org/10.1145/2494091.2494185

[15] Jan Humble, Andy Crabtree, Terry Hemmings, Karl-Petter Åkesson, Boriana Koleva, Tom Rodden, and Par Hansson. 2003. "Playing with the Bits" User-configuration of Ubiquitous Domestic Environments. In *Proceedings of the 5th International Conference on Ubiquitous Computing (UbiComp 2003)*. DOI: http://dx.doi.org/10.1007/978-3-540-39653-6_20

[16] Jisoo Lee, Luis Garduño, Erin Walker, and Winslow Burleson. 2013. A Tangible Programming Tool for Creation of Context-Aware Applications . In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2013)*. 391–400. DOI: http://dx.doi.org/10.1145/2493432.2493483

[17] Sarah Mennicken, Jo Vermeulen, and Elaine Huang. 2014. From Today's Augmented Houses to Tomorrow's Smart Homes: New Directions for Home Automation Research. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2014)*. 105–115. DOI: http://dx.doi.org/10.1145/2632048.2636076

[18] 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), 759–766. DOI: http://dx.doi.org/10.1016/j.sysarc.2013.03.015

[19] Khai Truong, Elaine Huang, and Gregory D. Abowd. 2004. CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. In *Proceedings of the 6th International Conference on Ubiquitous Computing (UbiComp 2004)*. 143–160. DOI: http://dx.doi.org/10.1007/978-3-540-30119-6_9

[20] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael Littman. 2014. Practical Trigger-Action Programming in the Smart Home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2014)*. 803–812. DOI: http://dx.doi.org/10.1145/2556288.2557420

[21] Bin Zhang, Pei-Luen Rau, and Gavriel Salvendy. 2009. Design and Evaluation of Smart Home User Interface: Effects of Age, Tasks and Intelligence Level. *Behaviour and Information Technology* 28, 3 (2009). DOI: http://dx.doi.org/10.1080/01449290701573978

Received January 2017; accepted May 2017