International Journal of Multimedia and Ubiquitous Engineering
Vol. 4, No. 3, July, 2009

SOEML: Smart Object Events Modeling Language based on Temporal Intervals

[1]Takuro Yonezawa, [2]Jin Nakazawa, [3]Goro Kunito, [4]Tomohiro Nagata, and [5]Hideyuki Tokuda
[1]*Graduate School of Media and Governance, Keio University*
[2]*Faculty of Environment and Information Studies, Keio University*
[3]*Research Laboratories, NTT DoCoMo, Inc*
[4]*Research Laboratories, NTT DoCoMo, Inc*
[5]*Faculty of Environment and Information Studies, Keio University*
takuro@ht.sfc.keio.ac.jp[1], jin@ht.sfc.keio.ac.jp[2],kunito@nttdocomo.co.jp[3], nagatat@nttdocomo.co.jp[4], hxt@ht.sfc.keio.ac.jp[5]

Abstract

This paper proposes a smart object event modeling language. By attaching sensor nodes to everyday objects, users can augment the objects digitally and apply the objects into various services. When creating such smart object services, users should define events, such as beverage of a cup turns cold or someone sits down on a chair, using physical values from sensors. The most common event definition for end-users is simply describing threshold of sensor values and boolean operation. When they want to define more complex events, such as multiple people sit down on chairs or a user starts to study using a pen and a notebook, they need to use a programming language. To define such complex event easily without complex programming language, we present a new event description language called SOEML based on temporal relation among simple events. We also provide users a visual interface to support users defining or reusing events easily.

Keywords: *SOEML, Smart Object Events Modeling Language, Temporal Intervals*

1. Introduction

To realize ubiquitous computing environment, technologies such as computer, sensor, and network has been improving. Especially, small sensor nodes equipped with various types of sensors such as thermometer, accelerator, or illuminometer have enormous potential to create context-aware services that assist a variety of human activities. Our life is filled with everyday objects, and we often have trouble with them (e.g. lost property). It is important to achieve the ubiquitous computing environment to apply everyday objects into ubiquitous services. Sensor nodes, when attached to everyday objects, enable us to gather real-world information as context. Recently, many researchers have been focusing on the ubiquitous services with these "smart objects"[4][9]. With smart objects, users will be able to enjoy the privilege of ubiquitous technology anytime anywhere in their lives.

To realize the smart object services in the home environment, the following two questions must be answered. The first is how to make our belongings smart. We already have a number of everyday objects. Therefore, providing an easy way to make them smart is important. We have addressed this with uAssociator[13], an easy association method between sensor nodes and objects. The second question, which this paper focuses on, is how to create smart object

services that reflect users' requirements. It is not practical to build all the services that users may want to use. Therefore an environment for users to create services themselves is necessary. Generally, users' requirements are occurred when an event happens in their surroundings. Since our surroundings are filled with many objects, detecting objects' events can be used for creating services that support our living. MediaCup[4] is a common example which uses object's event: *"when a MediaCup recognizes that the beverage in the cup is getting cold, MediaCup notifies a user to drink it quickly."* To create such services (as we say context-aware services) by user themselves, they must define objects' events by using sensors. However, since defining events is unfamiliar tasks for users, supporting users to define smart object events is necessary.

In this paper, we present a Smart Object Event Modeling Language called SOEML. SOEML is an XML-format language and it has following three features;

i) It enables users to define complex event by combining simple events, which is based on sensor value's thresholds, with considering temporal relationship.

ii) It enhances reusability of events by separating logic of events and target smart objects.

iii) It supports defining flexible events which are composed of multiple smart objects.

To define smart object events, considering temporal relationship between simple events is important. For example, "beverage turned cold" cannot be defined simply as "if the cup's temperature <= 40 degrees Celsius." This is because "beverage turned cold" implicitly means "the beverage which was hot turned cold." Therefore, the event should be defined as "if the cup's temperature >40 degrees Celsius and then <=40 degrees Celsius." This means that it is important for context definitions to describe temporal relation between simple events. Though there has been an increasing effort and interest in developing infrastructures or toolkits for defining context-aware applications [7][8][14], those toolkits only support simple boolean logic or limited temporal operation. We extended the Allen's interval-based temporal operation to provide fully function for modeling various types of events and designed definition method which is friendly with smart object events. Additionally, since it is burden task for users to define all events from scratch, defined event model should be reusable and should not be tightly coupled with smart object. However, components of pre-defined event such as thresholds of sensor values or event correlation logic may different in different objects or environments. Therefore, for providing reusability to event definition, it is important for users to understand pre-defined event easily and to provide independency between event model and its target object.

The rest of this paper is organized as follows. In Section 2, we discuss event correlating model for smart object events. In Section 3, we present details of SOEML. In Section 4, we present implementation of SOEML and toolkit for supporting to define SOEML. In Section 5, we present user study of SOEML. Then, we survey on related work in Section 6. Finally, we conclude the paper.

2. Event model

In this section, we discuss event model for smart objects. First, we describe sample events, and then we indicate requirements for the model of smart object events.

2.1 Context with smart objects

Context-aware applications are typically difficult to build since the developer has to deal with a various issues related to the sensing, representing, aggregating, storing, or reasoning of context. Of those, how to install sensor technology to users' home is a first obstacle. Professional installation will continue to be appropriate for the critical classes of applications - security systems, life support, dangerous automation or actuation - and for institutions that can afford technicians. However, this approach is not currently feasible for non-critical, everyday applications, of a professional installation may outweigh an application's perceived value [13]. Since it is easy to attach sensor node to users' belongings, using sensor-attached objects, what we call smart objects, is one of available ways for creating context-aware applications at home. Since our life is filled with many objects, by monitoring objects' status, various contexts can be detected. Table 1 shows perceptible context with smart objects. Since users are a domain expert of their own home, there are many other contexts or applications better tailored to their needs or preferences. Though each context is definitely recognized correctly by professional systems such as cooking [17] or brushing teeth [16], moderate context with smart objects also must be useful in everyday, non-critical applications. In the real world, states of objects change every now and then. Users must define a part of this change in state as a context, a trigger of certain services. In order to assist users establishing services, modeling the change of state should be simple. Therefore, we focus to define context by using smart objects, especially smart object events which is a certain change of the objects' status, by correlating simple thresholds of sensor values. In below, we show examples of how smart object events can be defined with sensor thresholds.

Table 1. Examples of contexts with smart object

Situation	Context	Available objects for recognize context
Morning	Waking up	pillow, bed
	Drinking[5]	cup
	Eating	spoon, fork
	Brushing teeth[18]	toothbrush
	Wearing	clothes
	Going bathroom	door, lavatory bow
	Preparing	bag
	Leaving home	door, shoes
Daytime	Starting work	chair
	Writing	pen
	Meeting	multiple chairs
	Reading	book
	Walking	shoes
Night	Coming home	door
	Watching TV	remote controller
	Cooking[19]	food chopper, skillet
	Cleaning	cleaner
	Playing	toy
	Listen music	headphone
	Sleeping	pillow, bed

1. Beverage turned cold: For example, "beverage turned cold" cannot be defined simply as "if the cup's temperature <= 40 degrees Celsius." This is because "beverage turned cold" implicitly means "the beverage which was hot turned cold." Therefore, the event should be defined as “if the cup's temperature >40 degrees Celsius and then <=40 degrees Celsius." This means that it is important for smart object event definitions to describe temporal relation between simple events.

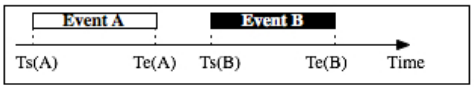
2. Meeting has started: An event "Multiple people sat down at the same time" can be useful for recognizing a "meeting" event. In this case, the event can be defined such as "if more than 3 chairs detect a movement by persons’ sitting at least once in 30 sec."

2.2 Event correlation

As we mentioned above, smart object events can be defined by correlating simple events that are using thresholds of sensor values. Therefore, providing suitable event correlation model is important for defining complex smart object events. In past work for supporting to define context [5][12][20] simple event correlation operators such as conjunction(AND) or disjunction(OR) operation are often used. These operators are widely used for it being intuitively understandable. However, in example cases above, it is impossible to meet user's demand only by setting threshold or simple operator. As shown in above examples, it is important to make consideration of temporal relation between simple events for defining more complex events. However, as complexity of the events increasing, defining the events which concerns with temporal concept is difficult task for users since it requires cumbersome states management or multi-thread programming.

In our event model, we apply Allen’s temporal relation logic for event correlation. Allen argues that all events have duration and considers intervals to be the basic time concept. A set of 13 relations between intervals is defined, and rules governing the composition of such relation controls temporal reasoning. By using 13 relations such as “before”, “overlaps” or ”during” (see Figure 2) , users can define temporal relation between events properly. For example, the event “beverage turns cold” can be expressed as “a cup’s temperature is less than or equals to 40 degrees Celsius meets it is greater than 40 degrees Celsius”. In addition, we extend the set of relations for defining flexible events. More correlation operators are deployed as part of application or middleware. In [10], correlating operators are classified into following categories: conjunction, disjunction, sequence, concurrency, negation, iteration, selection or aggregation. For defining various smart object events, event modeling language should be able to express these correlations. As mentioned in [10], a part of these correlations can be expressed with the set of 13 relations. For example, [12] defines correlating operator “sequence” as following: sequence of two events E1 and E2 occurs when E2 occurs provided E1 has already occurred. This operator’s meaning can be expressed with combination of Allen’s temporal relation such as (E1 before E2) or (E1 meets E2). However, some correlation cannot be defined only by Allen’s relation or simple boolean operation.

For defining various events, we add a new relation called “any” to a set of Allen’s temporal relation. The relation "any" is a special temporal relation that fires an event when all related events fires in any relation. In addition, “any” correlation enables users to define flexible events such as “more than 3 events are fired” by defining fired event number and its condition. For example, “meeting has started” can be expressed as Figure 3. This enables users to define various event correlations with unified fashion. This, in turn, improves easiness of defining various events.



Relation Name	Interval Relation	Timestamps Relation
before		Te(A) < Ts(B)
meets		Te(A) = Ts(B)
overlaps		Ts(A) < Ts(B) < Te(A) < Te(B)
ended by		Ts(A) < Ts(B) < Te(A) = Te(B)
contains		Ts(A) < Ts(B) < Te(B) < Te(A)
starts		Ts(A) = Ts(B) < Te(A) < Te(B)
equals		Ts(A) = Ts(B) < Te(A) = Te(B)
started by		Ts(A) = Ts(B) < Te(B) < Te(A)
during		Ts(B) < Ts(A) < Te(B) < Te(B)
ends		Ts(B) < Ts(A) < Te(A) = Te(B)
overlapped by		Ts(B) < Ts(A) < Te(B) < Te(A)
met by		Te(B) = Ts(A)
after		Te(B) < Ts(A)

Figure 2. Allen's temporal interval logic

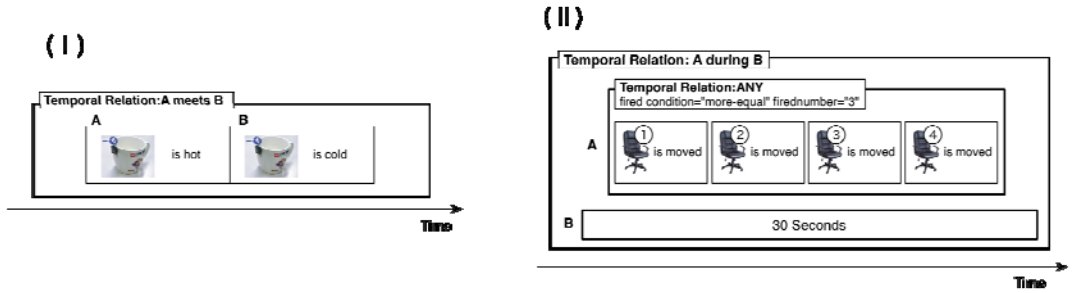


Figure 3. Event definition with temporal relations:
(I) beverage turns cold, and (II) meeting has started

3. SOEML: Smart Object Event Modeling Language

In this section, we present details of SOEML. With SOEML, users can define both simple event and complex event easily. First, we show requirements for defining smart object events. Then, we present the details of SOEML.

3.1 Design requirements

In the section 2, we argued that event model for smart object should be able to define complex event by correlating primitive events taking temporal relation into consideration. In addition, there remain other requirements for modeling smart object events. Since it is burden task for users to define all events from scratch, defined event model should be reusable and should not be tightly coupled with smart object. These observations lead us to the following design concept.

1. Simplicity and expressiveness: In order to assist developers establishing services, modeling the change of smart object state should be both simple and expressive. Not only simple event correlation such as boolean logic but also temporal relation or other correlation should be able to be expressed. For that purpose, we extend Allen's temporal logic for event correlation.

2. Readability and portability: For facilitating reusablity of events, modeling language should have readability and portability. Though various middleware or database research supports event correlation function, the subscription language is often a subset of SQL or mathematical event algebra, which is not sufficiently expressive. For providing readability and portability, we develop XML-based event modeling language.

3. Independency of model and its target object: Different person has different objects. However, objects which are classified into same category should be able to share same events model. For adapting one event model to many objects, we provide independency of event model and its target object.

3.2 Structure

SOEML is composed of 5 main elements: <event_template>, <temporal_relation>, <time_duration>, <smart_object> and <event>. Figure 4 shows an overview of these 5 elements. SOEML is composed of the basic element <event_template>. And the element <event> can be defined by correlating <event_template> and <smart_object>. The both elements <event_template> and <event> have two types: Atomic and Composite. Composite type is defined by correlating two or more Atomic type with temporal logic. Details of each element are shown below in order.

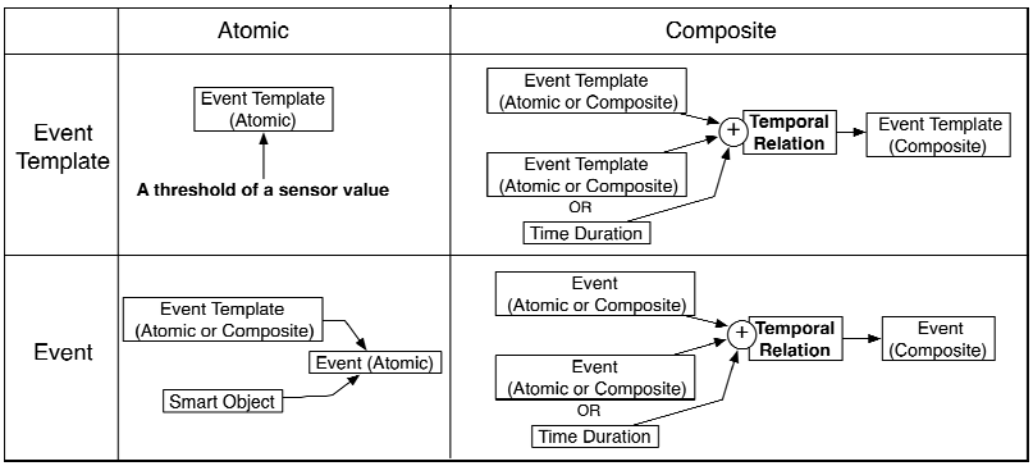


Figure 4. Overview of elements in SOEML

3.2.1 Event Template and Temporal Relation

Atomic Event Template

Event Template is a basic event model in SOEML. Smart object events can be detected by adapting the Event Template to target smart objects. There are two types of the Event Template: Atomic and Composite. Atomic Event Template is used for defining primitive events, using a threshold of a sensor value. Figure 5 illustrates an example of Atomic Event

Template which defines "temperature is less than 30 degrees." For describing both Atomic and Composite Event Template, <event_template> element is used. The "type" in <event_template> is used for specifying whether the event template is Atomic or Composite. In element <event_template type="atomic">, users describe concrete sensor information. An element <node_condition> contains 3 elements: <type>, <sensor> and <value>. <type> is used for specifying sensor node types such as uPart[3] or Mote[15]. The attribute <sensor> specifies which sensor on the node should be monitored. In case of using uPart sensor node, there are 3 types of sensors; temperature, movement and illuminance. The attribute <value> is used for setting a certain threshold of sensor values. An attribute "exp" in <value> governs types of thresholds. We prepared the following 7 types of thresholds; greater, greater-equal, equal, less, less-equal, between and except.

```
<event_template type="atomic" name="SomethingCold">
    <node_condition>
        <type>uPart</type>
        <sensor>temperature</sensor>
        <value exp="less">30</value>
    </node_condition>
</event_template>
```

Figure 5. An example of atomic event template

Composite Event Template

Composite Event Template is a complex event which is defined by the correction of two event templates. To combine multiple event templates, we use Allen's temporal interval logic. We use a set of 13 relations between intervals which are defined in [2] and a new relation "any" which we added to improve SOEML's flexibility.

Figure 6 shows an example of the Composite Event Template. This template relates two Atomic Event Templates by *"meets"* relation. Element <temporal_relation>, which has two or more <event_template> elements, is used for defining temporal relationship between event templates. It has a *"type"* attribute which defines a temporal relation from 14 types of relation. In this case, *"type"* is set to *"meets"* relation. Atomic Event Template *"SomethingHot"* in *<temporal_relation>* is defined as "temperature >= 30." The Atomic Event Template named "SomethingCold" is, as shown in Figure 5, defined as "temperature < 30." Therefore, this Composite Event Template means that the temperature of something went down from over 30 degrees to below 30. As shown in Figure, defined elements can be referred by "ref" attribute. Additionally, if users set an attribute "condition=false" with "ref" attribute, it means negation of the defined event template.

```
<event_template type="composite"  name="SomethingTurnsCold">
    <temporal_relation type="meets">
        <event_template ref="SomethingHot"/>
        <event_template ref="SomethingCold"/>
    </temporal_relation>
</event_template>
```

Figure 6. An example of composite event template

3.2.2 Time duration

The element <time_duration> is used for defining certain time interval. Take the event *"the temperature of a target is below 30 degrees for 10 seconds"* for example. In this case, users can define the event as shown in Figure 7. The element <time_duration> can contain time intervals such as <hour>, <min>, <sec> and <millisec>. The event above fires when *"SomethingCold"* event continually occur for 10 seconds. If the *"type"* attribute of <temporal_relation> is set to *"contains"* relation, the event fires when *"SomethingCold"* event starts and finishes within 10 seconds.

```
<event_template type="composite"  name="SomethingColdOver10Sec">
        <temporal_relation type="during">
                <time_duration>
                        <sec>10</sec>
                </time_duration>
                 <event_template ref="SomethingCold"/>
        </temporal_relation>
</event_template>
```

Figure 7. An example of time duration

3.2.3 Event and Smart Object

An Event Template turns to a concrete Event when the user pairs it with a smart object. Figure 8 shows an example of an event named *"CupTurnsCold."* The element <smart_object> contains two kinds of information; sensor information such as node types and node ID, and object information such as object name or owner of the object.

```
<event type="atomic" name="CupTurnsCold">
        <smart_object ref="Cup"/>
        <event_template ref="SomethingTurnsCold"/>
</event>

<smart_object>
        <sensor_info>
                <type>uPart</type>
                <id>1.2.3.4.0.1.0.12</id>
        </sensor_info>
        <object_info>
                <name>Cup</name>
                <owner>Takuro Yonezawa</owner>
        </object_info>
</smart_object>
```

Figure 8. An example of event and smart object

Alike Event Template, Event has both atomic and composite type. Atomic Event is used for defining events of a smart object. On the contrary, Composite Event is used for defining events which involves multiple smart objects. Figure 9 shows a Composite Event which defines *"If both a pen and a notebook are moved at least once within certain 10 second."* A Composite Event named *"Studying"* has a nested construction. First, it relates a Composite Event named *"PenAndNotebookMoved"* and time duration *"10 seconds"* with a *"during"*

relation. This means that when the event *"PenAndNotebookMoved"* occurs within 10 seconds, the event *"Studying"* fires. The event *"PenAndNotebookMoved"* is also a Composite Event: it relates the event *"PenMoved"* and *"NotebookMoved"* by a relation *"any"*. The relation *"any"* is a special temporal relation that fires an event when all inner events fires in any relation. Moreover, when *"any"* is used, additional attributes *"firedCondition"* and *"firedNumber"* can be set to the element <temporal_relation> for defining flexible conditions. For example, the code <temporal_relation type="any" firedCondition="more-equal" firedNumber="1"> means that when one or more inner events fire, the condition is fulfilled. There are 5 types of "firedCondition" - more, more-equal, equal, less-equal and less. By using *"firedCondition"* and *"firedNumber"*, logical addition or exclusive disjunction can be defined.

```
<event type="composite" name="Studying">
        <temporal_relation type="during">
                <event type="composite" name="PenAndNotebookMoved">
                        <temporal_relation type="any">
                                <event ref="PenMoved"/>
                                <event ref="NotebookMoved"/>
                        </temporal_relation>
                </event>
                <time_duration>
                        <sec>10</sec>
                </time_duration>
        </temporal_relation>
</event>
```

Figure 9. An example of composite event template

3.3 Discussion

We discuss features of SOEML in terms of descriptive capability and reusability.

3.3.1 Description capability

SOEML enables users to define complex events by correlating simple events which is based on thresholds of sensor values. Because the correlation of events can be defined by 14 temporal interval logics including Allen's 13 relations and the "any" relation that we propose, users can create various smart object events without using complex programming language. Let us get back to the example *"if more than 3 chairs detect a person's weight at least once in 30 sec"* in Section 2. Though this event is difficult to define with programming language, users can model the events by SOEML simply as Figure 10. The event *"Meeting"* correlates *"MoreThan3ChairsUsed"* event and time duration *"30 seconds"* by a *"during"* relation. *"MoreThan3ChairsUsed"* event is also a Composite Event which uses the temporal relation *"any"*. In this case, *"firedCondition"* and *"firedNumber"* is set for defining the condition of *"more than 3 chairs being used."* With *"any"* relation, users can define events flexibly.

3.3.2 Reusability of events

In SOEML, an event is expressed with a pair of Event Template and Smart Object. In other words, when users want to adapt the same event recognition logic to an object, the only thing users need to do is to change the Smart Object paired to the Event Template. This feature

boosts up the reusability of SOEML. Additionally, all events are based on simple thresholds of sensor values. This reduces difficulty for users to modify event defined using SOEML by other people. We basically do not assume that the users can use the same event that they have modeled in a different environment. This is because the sensor values that the object detects differs when the environment of user changes. Therefore, the feature that a user can use or easily modify an event which another person has already made is very important. Additionally, we provide a toolkit to support users to reuse smart object events. In the next section, we show the details of toolkit.

```
<event type="composite" name="Meeting">
    <temporal_relation type="during">
        <event type="composite"  name="MoreThan3ChairsUsed">
            <temporal_relation type="any"  firedCondition="more" firedNumber="3">
                <event ref="Chair1Used"/>
                <event ref="Chair2Used"/>
                <event ref="Chair3Used”/>
            <event ref="Chair4Used"/>
            <event ref="Chair5Used"/>
        </temporal_relation>
    </event>
    <time_duration>
        <sec>30</sec>
    </time_duration>
    </temporal_relation>
</event>
```

Figure 10. An example of event “meeting”

4. Implementation

In this section, we describe implementation of SOEML and toolkit for supporting to define events with SOEML.

4.1 Implementation of SOEML

We implemented SOEML schema, parser and event detector with Java and JAXB[1]. We also defined some events in Table 1. As examples, we show “leaving home” and “coming home” events in Appendix. Figure 11 shows overview architecture of SOEML event detector. As sensor nodes, current implementation of SOEML supports uPart[3], Mote[14] and SunSpot[15]. We assume that the users will use smart object services in the home environment where there are one or more computers which operate applications, cooperating with sensor nodes mounted on objects. Events which can be detected are highly influenced by temporal intervals, which sensor nodes send packets to a computer. If the intervals of the packets being sent are different, it is impossible to evaluate whether the composite event occurred or not. Therefore, after getting sensor data, timing resolver in Figure 11 slides all sensor data into the same time intervals. We set this interval to 500 milliseconds.

The current detection mechanism for temporal evaluation is based on an event-driven algorithm. When an event constructing a Composite Event is recognized, a temporal evaluation is executed every interval (500 milliseconds). Each event stores both the start time and the end time, and the system computes every past events or time duration which users have defined. Note that the start time and the end time of Composite Event or Atomic Event

which contains Composite Event Templates are dependent to the inner events which construct the Composite Event/Event Template. Figure 12 shows the time chart of the Composite Event Template *"SomethingTurnsCold"* shown in Section 3. The Event Template *"SomethingHot"* started from T1 and ended at T2, while *"SomethingCold"* started from T2 and ended T3. This temporal order matches *"meets"* relation, so the Composite EventTemplate *"SomethingTurnsCold"* fires. In this case, while the Atomic Event Template *"SomethingHot"* and *"SomethingCold"* fires every time when the sensor data matches the event, *"SomethingTurnsCold"* fires at T2 when *"SomethingHot"* and *"SomethingCold"* matches *"meets"* relation. If *"SomethingTurnsCold"* is a part of upper Composite Event Template, *"SomethingTurnsCold"* is treated as an event which has time interval from T1 to T2.

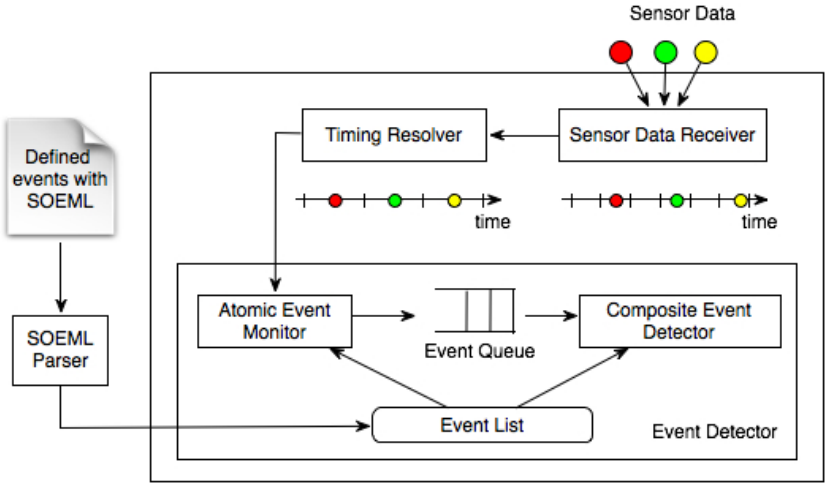


Figure 11. System architecture of SOEML event detector

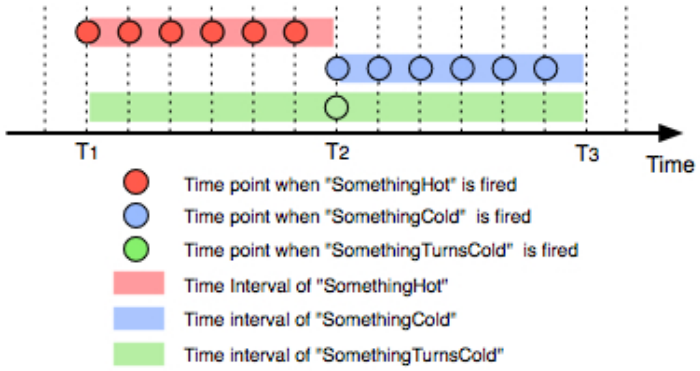


Figure 12. Time chart of event template “somethingTurnsCold”

4.2 Tools for supporting to define events using SOEML

We also implemented toolkit for defining events using SOEML easily. This toolkit has following 2 functions: 1) visual interface for defining and managing events model with SOEML and 2) intuitive tool for installing new smart object to SOEML.

Visual interface

We implemented a prototype of user interface for describing and managing events using SOEML (see Figure 13). By using the interface, users can load SOEML, visualize its structure, edit and save the events as XML code. The interface provides a structure of the elements visually with animation. Users can define SOEML components such as atomic event or event template, composite event or event template with registered smart objects by using this interface (e.g., defining appropriate element by using combobox). As common problem running through all of hierarchical description method, it is difficult to manage elements as the number of them grows. This problem also causes a detrimental effect to scalability of SOEML. In order to reduce this matter, we implemented a prototype tool for managing events. Since the tool employs Zooming User Interface [20], users can manage events and understand even deep hierarchical events intuitively.

Smart object installation tool

The element <smart_object> contains two kinds of information; sensor information such as node types and node ID, and object information such as object names and owner of the object. Applications, which provide smart objects services, need to know what object each sensor node is monitoring. This, in turn, requires association, or making a semantic relationship between the sensor node ID and its object information. However, specifying sensor ID and entering information related to objects are burden task. For solving this problem and defining <smart_object> element easily, we adapt uAssociator which we developed in [13]. It provides an easy association method (only spotlighting and capturing an object with a sensor node with special camera unit) and the associating information is stored into a JPEG file in an XML format (see Figure 14). The element <smart_object> contains the associating information. By copying this information to SOEML or selecting the JPEG file through a user interface shown in above, users can define and use <smart_object> easily.

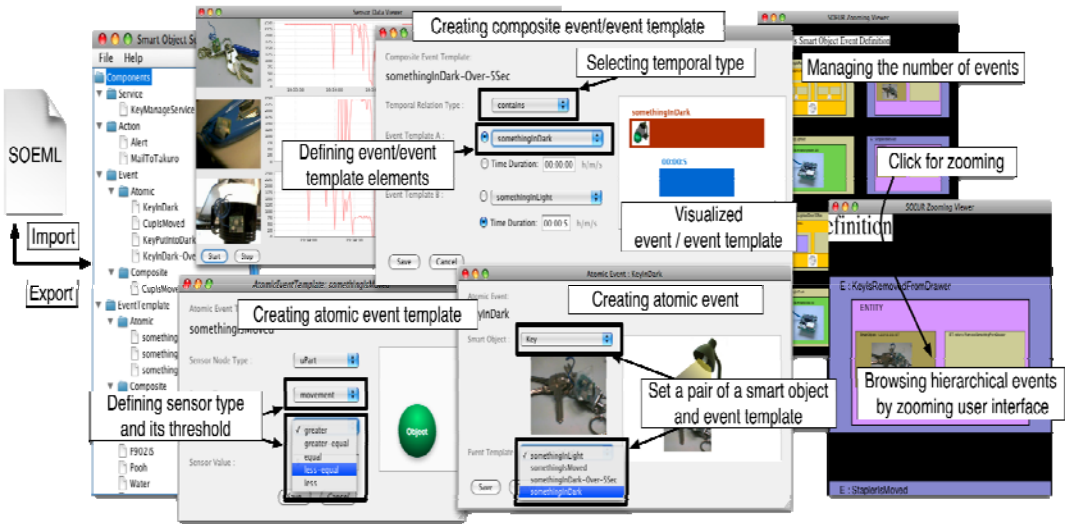


Figure 13. User interface for editing SOEML

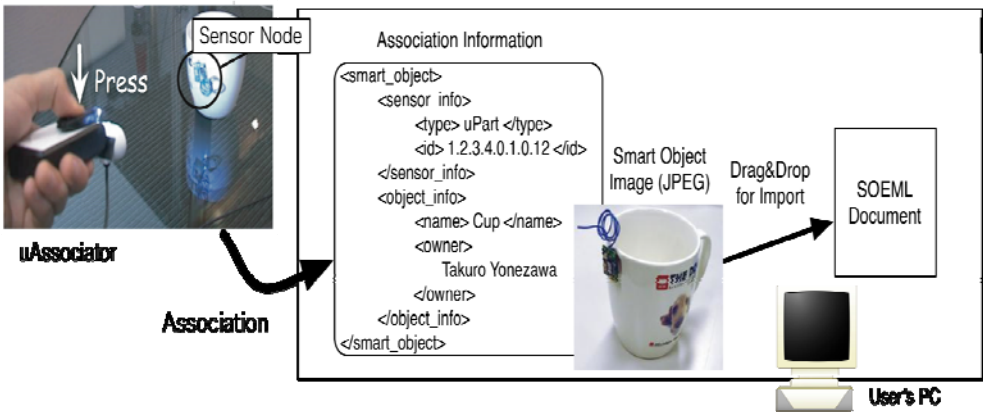


Figure 14. Tool for associating sensor node and everyday object

5. User study

This section evaluates easiness of modeling events and reusability of SOEML through a user study.

5.1 Study methodology

For evaluating efficiency of SOEML, we tested how users leverage SOEML itself. Therefore, we asked users to define some events with SOEML by only using text editor, not by richer interface we built. The participants are 8 individuals (6 Male, 2 Female, age range 19-35) who are all novice users of SOEML. Participants A-F are familiar with programming (over 5 years of experience), and participants G-H are not familiar with programming (less than a half year of experience). We ask them to define following 4 events of objects (key, stapler and pen attached with uParts) :

Event 1: Light value of the key is under 100.

Event 2: Light value of the key is over 100, and then turns to under 100.

Event 3: Light value of the key is over 100, and then continues under 100 over 5 seconds.

Event 4: Light values of the 3 objects are over 100, and at the same time (within 5 seconds) the light values turn to under 100.

Before participants defined events with SOEML, we gave a 15 minutes tutorial of how to use SOEML. We also prepared a template of SOEML. It includes <smart_object> description which indicates 3 objects and a few samples of <event_template> and <event> description (see Figure 15). After the experiment, we had a questionnaire about usability of SOEML. Additionally, we ask participants A-F to define same events with Java to compare easiness and reusability against SOEML. For java programming, we also prepared a template program which already has a function to get sensor data from each sensor node on key, stapler and pen. This allows participants to focus only on program logics to define events.

```
<event type="composite" name="CupTurnsCold">
  <event_template ref="somethingTurnsCold"/>
  <smart_object ref="cup"
</event>

<event_template type="composite"
name="somethingTurnsCold">
  <temporal_relation type="meets">
    <event_template ref="somethingIsHot"/>
    <event_template ref="somethingIsCold"/>
  </temporal_relation>
</event_template>

<event_template type="atomic" name="somethingIsHot">
  <node_condition>
   <type>uPart</type>
   <sensor>temperature</sensor>
   <value exp="greater-equal">30</value>
  </node_condition>
</event_template>
```

```
<event_template type="atomic" name="somethingIsCold">
  <node_condition>
   <type>uPart</type>
   <sensor>temperature</sensor>
   <value exp="less">30</value>
  </node_condition>
</event_template>

<smart_object>
  <sensor_info>
    <type>uPart</type>
    <id>1.2.3.4.0.1.0.242</id>
  </sensor_info>
  <object_info>
    <name>Cup</name>
  </object_info>
</smart_object>
….
```

Figure 15. Template of SOEML

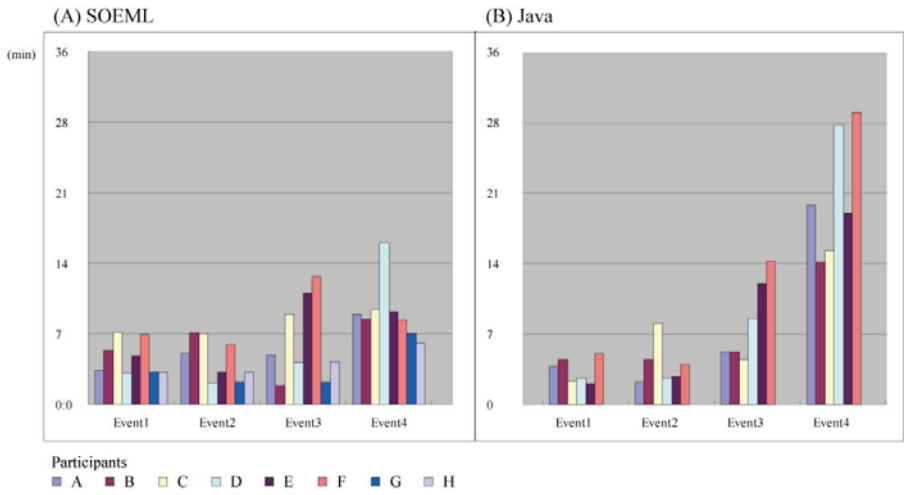


Figure 16. Experiment result of each participant (left: SOEML, right: Java)

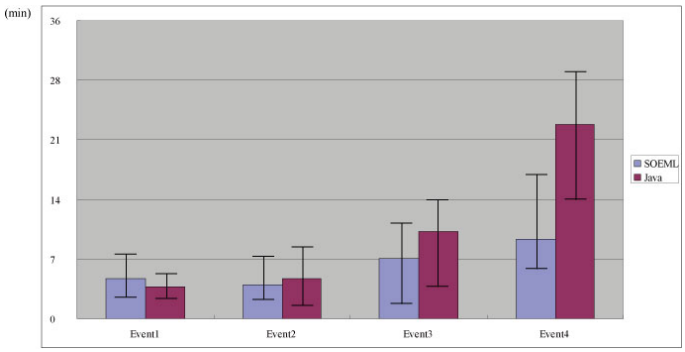


Figure 17. Average time for defining each event

5.2 Study result

Figure16 illustrates the time participants took to define each event. Average time of each definition is also shown in Figure 17. We discuss the result of the study from two points of view: easiness of modeling events, and reusability of events.

5.2.1 Easiness of modeling events

With a short tutorial, all participants succeeded to define all events correctly. In addition, the experiment show that SOEML improved in efficiency in complex events compared to simple events (see Figure 16). By comparing time taken to define events between programming expert (participants A-F) and non-expert (participants G-H), SOEML seemed to require no high programming skills. Participants who defined events with both SOEML and Java claimed that programming relating time interval requires strict state management, which makes SOEML much easier to define events compared to Java. They also assumed that by getting used to using SOEML, they would be able to define events more quickly. Furthermore, some participants indicated that since SOEML has simple policy to define events, users can focus on modeling events without frustration concerning states or flow management and on testing accuracy of temporal logic or thresholds of sensor values. However, error check of SOEML parser was not smart enough to check all the spelling errors, which lead to participants spending more time on searching for spelling errors. It is said that there are lots of events relating to time interval, so Graphical User Interface should be needed to define events more intuitively. We confirmed the effectiveness of defining complex events with SOEML especially relating to time interval.

5.2.1 Reusability of events

```
<event type=”composite”
name=”AllIsNotIlluminatedDuring5Sec”>
   <temporal_relation type=”during”>
        <event ref=”AllIsNotIlluminated”/>
        <time_duration><sec>5</sec></time_duration>
   </temporal_relation>
</event>

<event type=”composite” name=”AllIsNotIlluminated”>
    <temporal_relation type=”any”>
         <event ref=”KeyIsNotIlluminated”/>
         <event ref=”StaplerIsNotIlluminated”/>
         <event ref=”CupIsNotIlluminated”/>
    </temporal_relation>
</event>
```

```
<event type="composite"
name="StaplerAndKeyAndCupBecomesDarkIn5Sec">
  <temporal_relation type="during">
    <event type="composite"
name="StaplerAndKeyAndCupBecomesDark">
        <temporal_relation type="any">
           <event ref="KeyBecomesDark"/>
           <event ref="StaplerBecomesDark"/>
           <event ref="CupBecomesDark"/>
        </temporal_relation>
    </event>
  </temporal_relation>
</event>
```

Figure 18. Comparison of Event 4 definition among different participant (Left: participants D, right: participant H)

In Java program, the program defining simple events were similar among participants; however, complex events were programmed completely different from others. For example, a participant used thread to time, while another attempted to time using the number of received sensor data packets. On the contrary, event definition using SOEML were almost identical. Figure 18 illustrates the SOEML definition of Event 4 from two participants. The way they name and capsulize events differ, nevertheless, the same logic is used to assemble, enabling

others to reuse the events somewhat easily. Furthermore, most participants used Event Template to define events of different smart objects. Participants arrogate that reusing already-defined events is painless, and events can be easily dilated. On the other hand, some claimed that to define complex events, large numbers of modules are needed, which makes users hard to manage all. This problem is often claimed in other structured hierarchical models. To reduce this problem, we learned the importance of interface to manage modules effortlessly.

6. Related work

Dey et. el. interviewed 20 people without programming skills about favorable application for ubiquitous computing environment [5] . As a result, 80% out of 371 proposed applications can be described with if-then rules. This indicates the efficiency of the if-then rule as an end user programming in the ubiquitous computing environment. Following these results, Dey et.el. have built a visual programming environment which enables users to write the if-then rule. However, the definable rules are limited to events possessing a simple operation such as boolean logic. Similarly, there have been many approaches to define services using the ECA(Event-Condition-Action) rule. Shankar et. el. quoted that the ECA rule could not assign conditions before and after firing which was a fault of ECA[8]. Shankar adopted the ECPAP rule which adds the before and after rule, and built Petri-net for plural rules, increasing the number of rules to be able to change. However, their target users who defines the ECPAP rule is an owner of the room, not the non-expert users. Jung focused on the users' mental model, proposing an event definition method using 5W1H[6]. However, to write in a 5W1H method, there must be an environment where the sensor data is able to be understood as context, therefore it is insufficient to use at home or non-instrumented environment.

Context toolkit[19] gathers the components prepared for each sensor nodes and acquires the multiple sensor nodes' information as input, which allows the user to understand complex situation. However, the availability of using multistage combination of components, for example, combination of a result of a collection and a component, is not mentioned. T.GU[20] categorized the real world's information and defined them using OWL, which enables the reasoning mechanism to autonomously interpret the circumstances. Users can determine a context by referring the OWL defining the context which has a trigger for the same situation. Nevertheless, it is only the definition of context that can be referred, and the result of situation evaluation cannot be used as the input when constructing a context.

Composite event detection research has been done in active database research. Yoneki et. el. proposed interval-based temporal operation to describe events in the sensor network systems [10]. They defined 10 types of composite event operators such as conjunction, disjunction or concatenation. Though their semantics covers various types of composite events (e.g. including location information), high level language for defining the events is unconsidered. On the contrary, the Tag and Think [7] is a research using temporal relationship to detect events alike ours. In Tag and Think, in order to estimate what the object is from the values from the sensor nodes attached to the object, developer defines the relation of the object's status and the possible status considering the temporal relationship, and evaluates from the state transition diagram and the status obtained by the experimented data. Tag and Think defines the object's status with the amount of change of a sensor data at a certain time, therefore it enables estimation with high accuracy to various environment. On the contrary, we have focused on not only the same object, but also on multiple objects using

the threshold of sensor value. Additionally, the statement of an *"any"* relation enables users to define events flexibly.

7. Conclusion

To enable context-aware services to fit user's life or respond to users' request, the importance of toolkit will be increased in the decade ahead. To realize toolkit for smart object services, this paper presents a smart object event modeling language called SOEML. Defining events is an unfamiliar task for users, so easy way of describing without complex programming is necessary. SOEML enables users to define both simple and complex event based on thresholds of sensor values by using XML format. It provides following three advantages to define smart object events: i) simplicity and expressiveness, ii) readability and portability, and iii) reusability. To define complex event, users can use simple 13 types of temporal logic, and new relation "any" which improves flexibility of event definition. We also confirmed advantages of SOEML through user study with our prototype in two dimensions: (1) easiness of modeling smart object events and (2) reusability of the defined events. Finally we describe future work. First is cooperating with other sensors such as location sensors or RFID. By introducing various sensors to SOEML, more complex event can be defined. Second is connecting events to various actuators. With cooperating information appliances, a toolkit which supports to bootstrap and create smart object services can be realized.

Acknowledgement

This work was supported by NICT as a part of the Dynamic Network Project.

References

[1] jaxb: JAXB Reference Implementation. https://jaxb.dev.java.net/.
[2] J. F. Allen. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832.843, 1983.
[3] M. Beigl, C. Decker, A. Krohn, T. Riedel, and T. Zimmer. uparts: Low cost sensor networks at scale. In Ubicomp Demo Session, 2005.
[4] M. Beigl, H.-W. Gellersen, and A. Schmidt. Mediacups: experience with design and use of computer-augmented everyday artifacts. Computer Networks (Amsterdam, Netherlands: 1999), 35(4):401.409, 2001.
[5] A. K. Dey, T. Sohn, S. Streng, and J. Kodama. icap: Interactive prototyping of context-aware applications. In K. P. Fishkin, B. Schiele, P. Nixon, and A. J. Quigley, editors, Pervasive, volume 3968 of Lecture Notes in Computer Science, pages 254.271. Springer, 2006.
[6] J.-Y. Jung, Y.-S. Hong, T.-W. Kim, and J. Park. Human-centered event description for ubiquitous service computing. In MUE, pages 1153.1157. IEEE Computer Society, 2007.
[7] T. Maekawa, Y. Yanagisawa, T. Hattori, and T. Okadome. A representation of objects for context awareness in ubiquitous environments. DBSJ Letters, 5(2):945.965, 2006.
[8] C. S. Shankar, A. Ranganathan, and R. Campbell. An eca-p policy-based framework for managing ubiquitous computing environments. In MOBIQUITOUS '05: Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, pages 33.44, Washington, DC, USA, 2005. IEEE Computer Society.
[9] K.-K. Yap, V. Srinivasan, and M. Motani. Max: human-centric search of the physical world. In SenSys' 05: Proceedings of the 3rd international conference on Embedded networked sensor systems, pages 166.179, NY, USA, 2005. ACM Press.
[10] E. Yoneki and J. Bacon. Unified semantics for event correlation over time and space in hybrid network environments. In R. Meersman, Z. Tari, M.-S. Hacid, J. Mylopoulos, B. Pernici, O¨ . Babaoglu, H.-A. Jacobsen, J. P. Loyall, M. Kifer, and S. Spaccapietra, editors, OTM Conferences (1), volume 3760 of Lecture Notes in Computer Science, pages 366.384. Springer, 2005.
[11] T. Yonezawa, H. Sakakibara, K. Koizumi, S. Miyajima, J. Nakazawa, K. Takashio, and H. Tokuda. upackage – a package to enable do-it-yourself style ubiquitous services with daily objects. In H. Ichikawa, W.-D. Cho, I. Satoh, and H. Y. Youn, editors, UCS, volume 4836 of Lecture Notes in Computer Science, pages 240.257. Springer, 2007.
[12] JooGeok Tan, Daqing Zhang, Xiaohang Wang, and HengSeng Cheng. Enhancing semantic spaces with event-driven context interpretation. In Pervasive, pages 80.97, 2005.

[13] C. Beckmann, S. Consolvo, and A. LaMarca. Some assembly required: Supporting end-user sensor installation in domestic ubiquitous computing environment. In International Conference on Ubiquitous Computing, pages 107.124, 2004.
[14]SunSpotWorld – Home of Project Sun SPOT by Sun Microsystems http://www.sunspotworld.com/
[15] Crossbow Technology: Low-power Wireless Mote Solutions Overview http://www.xbow.com/Products/wproductsoverview.aspx
[16] Y. Chang, J. Lo, C. Huang, N. Hsu, H. Chu H. Wang and P. Chi and Y. Hsieh. Playful toothbrush: ubicomp technology for teaching tooth brushing to kindergarten children. CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, pages 363-372, 2008
[17] Tran, Quan T. Mynatt, Elizabeth D. What Was I Cooking? Towards Deja Vu Displays of Everyday Memory. GVU Technical Report;GIT-GVU-03-33
[18] Bederson, B., Grosjean, J., Meyer, J. Toolkit Design for Interactive Structured Graphics
IEEE Transactions on Software Engineering(TSE). HCIL-2003-01, CS-TR-4432, UMIACS-TR-2003-03
[19] A.K. Dey, G.D. Abowd, and D. Salber, “A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications,” Human Computer Interaction, vol.16, no.2,3,&4, pp.97.166, 2001.
[20] T. Gu, H.K. Pung, and D.Q. Zhang, “A service oriented middleware for building context-aware services,” J. Network and Computer Applications, vol.28, no. 1, pp. 45-55, 2002

Appendix

Examples of smart object events

```
<event type="composite" name="leaving home">
 <temporal_relation type="during">
  <event type="composite" name="shoesMovedThenDoorMoved">
    <temporal_relation type="before">
      <event ref="shoesMoved"/>
      <event ref="doorMoved"/>
    </tempora_relation>
  </event>
  <time_duration><sec>20</sec></time_duration>
</event>

<event type="composite" name="coming home">
 <temporal_relation type="during">
   <event
   type="composite" name="doorMovedThenShoesNotMoved">
     <temporal_relation type="before">
         <event ref="doorMoved"/>
         <event ref="shoesIsNotMovedOver10Sec"/>
     </temporal_relation>
   </event>
   <time_duration><sec>30</sec></time_duration>
 </temporal_relation>
</event>

<event type=”atomic” name=”doorMoved”>
  <event_template ref=”somethingIsMoved”/>
  <smart_object ref=”door”/>
</event>

<event type=”atomic” name=”shoesMoved”>
  <event_template ref=”somethingIsMoved”/>
  <smart_object ref=”shoes”/>
</event>

<event type=”atomic” name=”shoesIsNotMovedOver10Sec”>
  <event_template ref=”somethingIsNotMovedOver10Sec”/>
  <smart_object ref=”shoes”/>
</event>

<event_template type="composite"
name="somethingIsNotMovedOver10Sec">
  <temporal_relation type="contains">
     <event_template ref="somethingIsNotUsed">
     <time_duration><sec>10</sec></time_duration>
  </temporal_relation>
</event_template>
```

```
<event_template type="atomic" name="somethingIsNotMoved">
  <node_condition>
     <type>uPart</type>
     <sensor>movement</sensor>
     <value exp="equal">0</value>
  </node_condition>
</event_template>

<event_template type="atomic" name="somethingIsMoved">
  <node_condition>
     <type>uPart</type>
     <sensor>movement</sensor>
     <value exp="greater">2</value>
  </node_condition>
</event_template>
```