Dynamic Ontology-Based Redefinition of Events Intended to Support the Communication of Complex Information in Ubiquitous Computing

Carlos Rodríguez-Domínguez, Kawtar Benghazi, Manuel Noguera, María Bermúdez-Edo, José Luis Garrido

Department of Computer Languages and Systems, University of Granada

Escuela Técnica Superior de Ingenierías Informática y Telecomunicaciones

C/ Periodista Daniel Saucedo Aranda S/N 18071 Granada, Spain

E-mail: {carlosrodriguez, benghazi, mnoguera, mbe, jgarrido}@ugr.es

Received: June 22, 2010 Accepted: August 31, 2010 DOI: 10.5296/npa.v2i3.421

Abstract

Ubiquitous systems should properly support the connection/disconnection of entities at run-time. Accordingly, the communication of information in this type of systems should be able to adapt themselves to changes in their structure and participant entities without any need of user intervention. In this regard and due to the dynamic nature of these systems, asynchronous communication is more useful than synchronous one. Particularly, the publish/subscribe paradigm is used as it supports, not only asynchronous communications, but also the loose coupling between system entities, which is an important requirement that has to be satisfied in order to deal with the changes in the configuration of the communications between the involved entities.

In this paper, we propose a model of dynamically redefinable events in order to support dynamic reconfiguration of communications in ubiquitous systems. We also introduce associated techniques for publishing, subscribing and combining those events. The structure of the events and their intended semantics will be formally specified in an ontology, which enables automated reasoning based on Description Logics.

Furthermore, the proposal is described by means of an example and implemented as part of a coordination middleware intended to support the development of ubiquitous systems.

Keywords: Event, Publish/Subscribe, Distributed Systems, Data Dissemination, Semantics, Ontology.

1. Introduction

Nowadays, it is very common to have ubiquitous systems [1][2] all around us. These systems are composed by mobile applications, services and "invisible" devices that seamlessly interact with the real environment while permanently being connected to a network (WAN, LAN, PAN, etc.). These systems also show the following requirements: (1) Several heterogeneous devices coexisting in the same network; (2) Limited availability of resources; (3) Continuous changes in the location of the participants; (4) Semantics of the exchanged information will depend on the network and the location. Traditional approaches do not fully satisfy all these requirements [3], as they are intended to support more general requirements and not those that are specifically associated with ubiquitous systems.

Asynchronous communications are used in order to deal with the dynamic nature of these systems, as new entities may appear or disappear constantly and, so, synchronous communications will have to be short-lived in most cases. Particularly, the publish/subscribe communication paradigm [4] is frequently chosen in order to exchange information in ubiquitous systems, mainly due to its asynchronous nature and the loosely coupling of communication participants [5].

Usually, information in the publish/subscribe communication paradigm is presented as *events*, which can be defined as *a significant occurrence that has a location in time and space* [6][7]. Representing information as events and adequately managing them may be useful in ubiquitous systems, as information is frequently exchanged only when the status of any of the participants has changed. For readability issues, all along the paper we will also refer to the message that notifies an event occurrence by the term event.

As new devices, services and applications are incorporated into ubiquitous systems (or removed from them), mainly due to steadily changes of their physical location, it becomes more and more complex to predict a suitable configuration to guarantee that, over time, each one will receive and process the required information so as to accomplish their intended objectives [8]. Thus, it is necessary to propose the definition of models and the usage of techniques that support the design and development of dynamically reconfigurable applications, services and devices, i.e., the ability to be modified and extended at run-time [9].

In recent years, some technological solutions based on the publish/subscribe paradigm have been proposed in order to support communications in ubiquitous computing systems. In this regard, some remarkable solutions are Mobile-Gaia [10], Lime [11] and MATE [12]. However, all these pieces of work do not deal with the conceptualization of the communicated information and its elements.

In this paper, we argue that a first step for providing communication participants with on-the-fly reconfiguration capabilities in ubiquitous systems is to include events that are able to be dynamically redefined in the publish/subscribe paradigm. Specifically, we will introduce the notion of *redefinable event*, a formal ontological model for these events, a formal adaptation of the publication and subscription techniques in order to support the proposed event model and a technique to combine events at run-time in order to build new ones and to notify them to interested entities.

This research work proposes the usage of formal semantic specification techniques in order to check the consistency of each subscription to events and to store the properties of the internal information that determines each event. Specifically, this formal specification will be represented in an ontology of events and related topics. Events will be able to be built by extracting parts of the internal information that will be stored in other events and by combining them in a consistent way (i.e., a combination of information properties that is consistent with the ontology). When this is done at run-time and new events are built from an event to which information is added or modified, we denote this fact as a *redefinition of events*.

The remainder of the paper is structured as follows. In Section 2, the proposal for dynamically redefining events in publish/subscribe communication paradigm is described. In Section 3 an overview of the technological support of the proposal is summarized. In Section 4 some work that is related to the proposal described in this paper is analyzed. Finally, Section 5 presents the conclusions drawn from this article and some directions for future work.

2. Redefining Events in Publish/Subscribe Paradigm

This section introduces the proposal to extend the publish/subscribe communication paradigm by providing events with dynamic redefinition capabilities. We present an event model, a definition of this model by means of an ontology, techniques for publishing and subscribing to these events and a technique to combine them at run-time. Finally, we describe an example scenario in order to give more insight about how the communication of information in ubiquitous systems can change at run time.

2.1 Redefinable Events

In the publish/subscribe communication paradigm, the information exchanged between communication participants (applications, services or devices) is encapsulated into *events*. We define an event as a communication element that is composed of a set of pieces of information that are related to some topics and can be produced by a communication participant as a result of a change in its state. These pieces of information are encapsulated into *event nodes*. An event node is structured as an *identifier-type-value* tuple. Each event may have an unlimited number of event nodes to represent any piece of information.

A valid identifier for an event node consists of a unique name. Primitive types can be any of the following ones: integer, floating-point, string, byte, boolean and sequences of these types. It is important to mention that an event node is also an event. This way, hierarchical structures (i.e., trees) can be expressed as events.

Each event is associated with one topic. Each topic represents the semantics of each event, which may be the result of the semantic combination of different topics. For example, if *temperature* and *noise* topics exist and there is a semantic association between both topics in the form of *comfort* topic, there may exist an event ε_i whose topic is *comfort* and whose event nodes are an association of the event nodes that are used to syntactically represent both *temperature* and *noise* events.

In order to deal with the dynamic nature of ubiquitous systems, we introduce the notion of *redefinable events* so as to allow adapting events to the changes that take place in the system (e.g., to add devices, to remove them, to modify the connections between the existing ones, etc.), by redefining the set of event nodes at run-time. Additionally, event topics can be dynamically inferred as follows:

- If an event is composed of one event node only, its topic will be the topic of such event node.
- Otherwise, if it is composed of more than one event node, its topic will be deduced from the semantic association between the topics of each event node.

Thus, in *redefinable events*, as event nodes may be dynamically added or removed, the topic of each event can be inferred. For example, if a redefinable event with a *temperature* event node is expanded by adding a *noise* event node, and both *temperature* and *noise* topics are semantically associated with the more general *comfort* topic, it will be inferred that the extended event is also semantically associated with the *comfort* topic.

The metamodel of events is shown in Figure 1 using UML class diagrams, and the elements of this metamodel are summarized in Table 1.

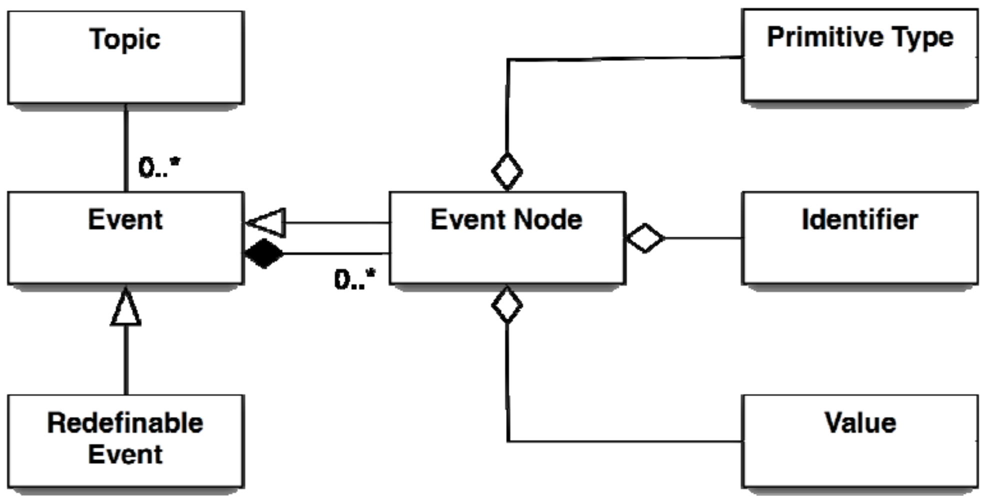


Figure 1. Redefinable event metamodel in UML.

Table 1. Event metamodel elements

Element	Interpretation
Event	Communication element that is composed of a set of pieces of information that are related to a specific topic
Event Node	Pieces of information structured as *identifier-type-value* tuples
Redefinable Event	Event whose structure could be redefined at run-time (i.e., adding, removing, or modifying event nodes)
Topic	Each topic defines a way to indirectly connect publishers with interested subscribers. Some examples of topics are temperature, humidity, users, devices, etc.
Relation Event-Topic	Each topic represents the formal semantics of each event
Relation Event-Event Node	An event is composed by a set of event nodes. Each event node has an associated topic, too. This way, it is possible to dynamically infer the topic of an event based only on the nodes: It will be deduced from the semantic association between the topics.
Relation Event-Redefinable Event	A Redefinable Event will be an specialization of an event that will be able to be modified at run-time. For example, it will be possible to add or to modify event nodes and to change the semantics associated with the original event.
Relation Topic-Topic	Each topic may be related with other topics as established by the ontology that represents the events (see Section 2.2).
Relation Event Node - Identifier - Primitive Type - Value	Each event node will have a tuple identifier-type-value associated.

2.2 An Ontological Model of Events

In Computer Science, an ontology is defined as "*a specification of a conceptualization*" [13]. An ontology describes the topics in a domain, the relations between them and the constraints on them. The metamodel shown in Figure 1 has been specified in a formal ontology. This ontological model is used in order to represent events, their structures (event nodes) and the semantic information associated with them. The ontology will also store the instantiations of the event model.

A system entity, which is going to be called the *semantic* entity, should monitor all changes the ontology may undergo and send out events so as to notify all the topics that may have been modified. Such changes could also be the result from automated reasoning procedures based on description logics [14] and that may infer implicit knowledge. Additionally, the *semantic* entity enables to check the substructure of a particular kind of topic from its ontological identifier.

Figure 2 shows a graphical representation of an instantiation of the ontology that is used in order to formalize topics and that is based on the metamodel of Figure 1. Note that in Figure 2, for a *comfort* type topic, the *semantic* entity will report that *user_condition* is a subtype of the *comfort* topic and that it is the result of the union of *noise* and *temperature* topics that, in turn, will have their own properties and characteristics.

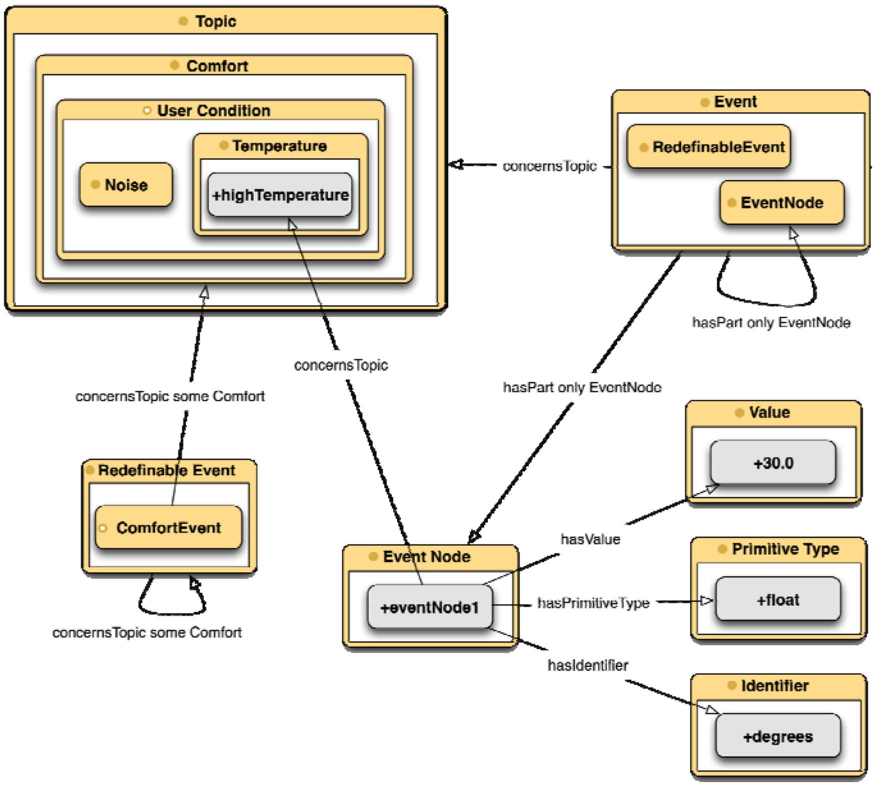
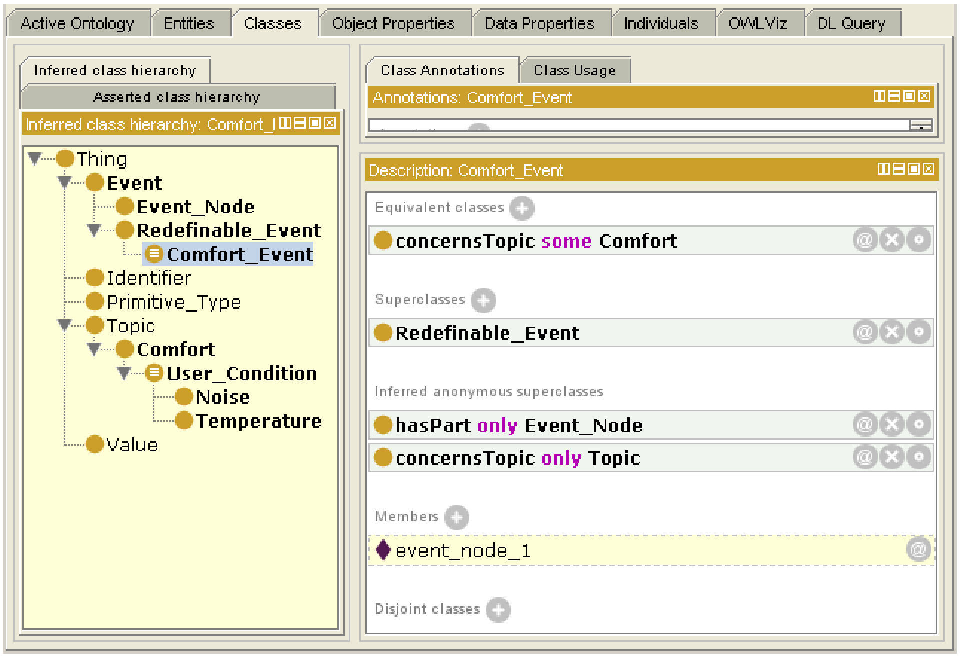


Figure 2. An instantiation sample of the ontology.

Figure 3 shows the implementation of part of the ontology in Protégé editor [15]. The ontology has been implemented in the OWL language [16], whose formal semantics is based on Description Logics.


Active Ontology | Entities | Classes | Object Properties | Data Properties | Individuals | OWLViz | DL Query

Inferred class hierarchy
Asserted class hierarchy
Inferred class hierarchy: Comfort_

```
Thing
  Event
    Event_Node
    Redefinable_Event
      Comfort_Event
  Identifier
  Primitive_Type
  Topic
    Comfort
      User_Condition
        Noise
        Temperature
  Value
```

Class Annotations | Class Usage
Annotations: Comfort_Event

Description: Comfort_Event
Equivalent classes
concernsTopic some Comfort

Superclasses
Redefinable_Event

Inferred anonymous superclasses
hasPart only Event_Node
concernsTopic only Topic

Members
event_node_1

Disjoint classes

Figure 3. An implementation of the ontology in Protégé ontology editor.

2.3 Event Subscription Techniques

A subscription σ to an event ε_i is defined as a *"filter over a portion of the event content (or the whole of it), expressed through a set of constraints"* [17]. The proposed events support two subscription variants:

- *Topic-Based*. Communication participants show interest in a topic and, from that moment, they receive events that are semantically related to that topic. For example, if a participant subscribes to the *comfort* topic, it will receive events, not only related to the *comfort* topic, but also to *temperature* and *noise* ones (if they are semantically related to it). Formally, the set of events that are received by a communication participant with a topic-based subscription σ_τ is defined as follows:

  Let *subscribe* be a function that filters all the events that are received by a participant on the basis of their associated topic, let T be the set of all semantically formalized topics and τ be a topic, then:

$$\sigma_\tau = subscribe(\tau), \tau \in T \qquad (1)$$

  Let E be the set of all possible events and let R be the product set (or the cartesian product) $E \times T$ in which all the relations (ε_i, τ) represent an event ε_i that is semantically related to the topic τ. We define the binary operation *"semantically related to"*, annotated as $\hookrightarrow$, as follows:

$$\varepsilon_i \hookrightarrow \tau \Leftrightarrow (\varepsilon_i, \tau) \in R,\ R = E \times T,\ \varepsilon_i \in E,\ \tau \in T \qquad (2)$$

  Let $S_{\sigma\tau}$ be the set of all the events received by a communication participant whose topic-based subscription is σ_τ, let $\varepsilon_i(k)$ be a function that retrieves the k-th event node of the event ε_i, which is also an event (see Section 2.1), and let n_i be the number of event nodes of ε_i, then:

$$S_{\sigma\tau} = \{\varepsilon_i \mid \varepsilon_i \hookrightarrow \tau\} \cup \{\varepsilon_i \mid \forall k \in \{1, \ldots, n_i\}, \varepsilon_i(k) \hookrightarrow \tau\} \qquad (3)$$

- *Content-Based*. Communication participants show interest in receiving events that are semantically associated with a topic when a set of constraints over the event nodes is accomplished. For example, it can be specified a subscription to the *temperature* topic and receive events only when this temperature is over 45º C. The set of events that are received by a communication participant with a content-based subscription $\sigma_{(\tau, P\tau)}$ is formally defined as follows:

  Let $\varepsilon_i(k)$ be the k-th event node of the event ε_i, let $t_{i,k}$ be the primitive type of the event node $\varepsilon_i(k)$, let a be a constant of any primitive type and t_a its primitive type. We define the binary operation *"is comparable to"* (annotated as †) as follows:

$$\varepsilon_i(k) \dagger a \Leftrightarrow t_{i,k} = t_a \qquad (4)$$

  If $\varepsilon_i(k) \dagger a$, five comparing operators can be defined:

$$\varepsilon_i(k) = a;\ \varepsilon_i(k) < a;\ \varepsilon_i(k) \leq a;\ \varepsilon_i(k) > a;\ \varepsilon_i(k) \geq a \tag{5}$$

These operators always follow a lexicographical order. For example, 4 < 5, “aaa” < “aab”, (4, 5, 6, 9) < (4, 6, 3, 1), etc.

Let s_j be a comparing operation between any $\varepsilon_i(k)$ and a constant a_i, where $\varepsilon_i(k) \dagger a$, let τ be a topic and $\varepsilon_i(k) \hookrightarrow \tau$. We define the predicate P_τ as follows:

$$P_\tau = s_1 \wedge \ldots \wedge s_m \tag{6}$$

Let *contentSubscribe* be a function that filters all the events that are received by a participant based on a topic τ and a set of constraints described by the predicate P_τ, then:

$$\sigma_{(\tau,P_\tau)} = contentSubscribe(\tau, P_\tau) \tag{7}$$

Let $S_{\sigma(\tau,\,P\tau)}$ be the set of all the received events by a communication participant whose content-based subscription is $\sigma_{(\tau,\,P\tau)}$, whose constraints are specified by the predicate P_τ, and let $v_{i,k}$ be the value associated with the event node $\varepsilon_i(k)$, then:

$$S_{\sigma_{(\tau,P_\tau)}} = \{\varepsilon_i : \varepsilon_i \in S_{\sigma_\tau}, \{P_\tau \wedge (\varepsilon_i(k) = v_{i,k})\} \vdash \neg\varnothing\} \tag{8}$$

In this formula, $\{P_\tau \wedge (\varepsilon_i(k) = v_{i,k})\} \vdash \neg\varnothing$ means that the predicate P_τ, when in conjunction with the equality $\varepsilon_i(k) = v_{i,k}$ has to be consistent (i.e., it not contains any logical contradictions). For example, if the predicate $P_{temperature} = \{\varepsilon_1(1) < 45\}$ and the event ε_1 is published with $\varepsilon_1(1) = 46$, then the subscriber will not receive the event, as $\{\varepsilon_1(1) < 45 \wedge \varepsilon_1(1) = 46\}$ is not a consistent set.

Each ubiquitous system should have an entity that will be in charge of storing every σ subscription. This entity will be known as the *subscription* entity and it will provide both *subscribe* and *contentSubscribe* functions. Additionally, it also provides the function *checkSubs*(ε_i), which checks whether a subscription related to the event ε_i exists or not. This function is the basis to publish events, as if an event should be published, it returns the identifiers of the participants that are interested in the specified event.

2.4 Event Publishing Techniques

An event ε_i must be sent out to a communication participant that is subscribed to the topic τ if that event is a member of the set $S_{\sigma\tau}$ or the set $S_{\sigma(\tau,\,P\tau)}$. For example, if a communication participant subscribes to the *comfort* topic and this topic results from the semantic combination between *temperature* and *noise* topics, then, *comfort*, *temperature* and *noise* events will be communicated to it. In order to solve the semantic relations between events and topics, the *semantic* entity is used.

Since each ubiquitous computing system has a *subscription* entity working, event publishers may invoke the function *checkSubs*(ε_i) to check whether they should publish an event ε_i or not. This way, events are only sent when a previous related subscription exists, thereby, avoiding unnecessary event publications. As it was specified in the previous section, the *checkSubs*(ε_i) will return the identifiers of the participants that are interested in the corresponding event. Hence, direct communications may be established between event publishers and subscribers.

It is important to note that the previous technique to publish events may be developed as a *publish* function in a centralized *publication* entity or as an intrinsic function to each communication participant. As a consequence, both centralized and distributed publication techniques are supported.

Moreover, when time constraints should be fulfilled in the ubiquitous system (for example, a restriction in the interval of time elapsed between a time of publishing an event and receiving it by its subscribers), it is recommended to choose the centralized publication technique, as it can be more predictable, which is an important requirement in real-time systems [18]. In case that the ubiquitous system should meet requirements like predictability and scalability, which would require on the one hand, the centralized technique and, on the other hand, the distributed technique, it will have to be decided which of the requirements are more important to be fulfilled. Also, these techniques may be mixed to meet some of these "conflicting" requirements by combining participants that use a *publication* entity and participants that implement their own publication techniques.

2.5 Event Combination

Event combination is made by using a *combination* entity, which will be in charge of receiving all the events that are notified in a system and will store the last received event of each topic. Whenever a new event is stored, it will extract some of its event nodes and will try to combine it with other nodes extracted from other stored events. In order to do that in a limited period of time, the *semantic* entity will be used so as to only make combinations that are consistent with the properties of the topics that are stored in the ontology. Additionally, event combinations in order to build events of certain topics will only be done if at least one system entity is interested in that topic (i.e., it is subscribed to that topic). For instance, if it exists a comfort topic whose properties are "degrees" and "db" and the *combination* entity receives several events related with several topics, it will only try to extract "degrees" and "db" information in order to build a new comfort event.

Whenever a new event is built, this entity will notify it and entities that are subscribed to its related topic will receive it.

By combining events with a *combination* entity, other entities will not have to take into account that the information that they require comes from different sources and has to be combined into a single piece of information.

2.6 An Example

In order to give more insight about how the structure of the information to be communicated in an ubiquitous system can be reconfigured at run time, we show a scenario that consists of a temperature sensor and a noise sensor (see Figure 4), to which is going to be added (A) and removed (B) a humidity sensor. These sensors publish temperature, noise and humidity events whenever values measured change. There is also a mobile application that is subscribed to the *comfort* topic, which is the most general one in this scenario. Its intention is to show to an end user all the information that is related with the comfort of the room in which it is situated.

It is important to note that the *combination* entity will receive all the events that are published in the whole system and will synchronously communicate with the *semantic* entity in order to request information or to provide it.

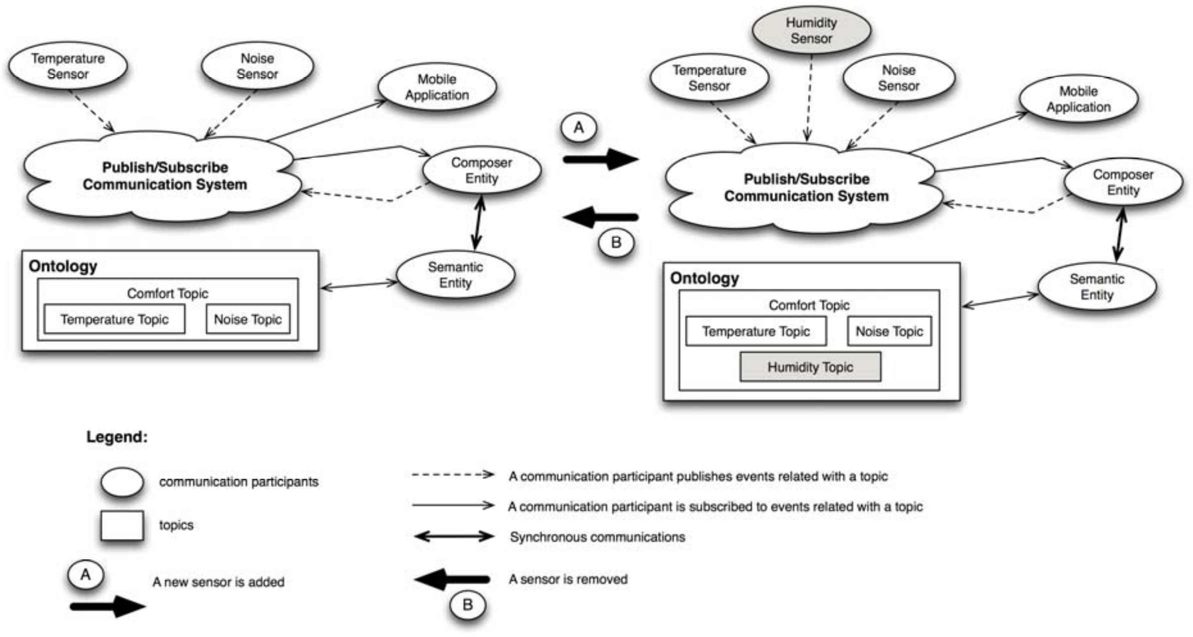

Temperature Sensor
Noise Sensor
Mobile Application
Publish/Subscribe Communication System
Composer Entity
Semantic Entity
Ontology
Comfort Topic
Temperature Topic
Noise Topic
A
B
Humidity Sensor
Humidity Topic
Legend:
communication participants
topics
A new sensor is added
A communication participant publishes events related with a topic
A communication participant is subscribed to events related with a topic
Synchronous communications
A sensor is removed

Figure 4. A proposed example scenario

Published events, which can be considered as sets of event nodes or tuples of *identifiers-types-values*, store the following information (where *x* is a constant of the same primitive type specified by the corresponding event node):

- $\varepsilon_{temperature} = \{(\text{“degrees”}, float, x)\}$
- $\varepsilon_{noise} = \{(\text{“db”}, integer, x)\}$
- $\varepsilon_{comfort} = \{\varepsilon_{temperature}, \varepsilon_{noise}\}$

The *combination* entity will receive both *temperature* and *noise* events and will try to build *comfort* events. In order to detect which are the consistent combinations between the retrieved pieces of information, this entity will use the *semantic* entity. Whenever it combines the information in an appropriate way, it will publish a *comfort* event. This event will be notified to the mobile application, which will use the internal information of it in order to fulfil its objective (i.e., to show the available comfort information of a room).

In this initial scenario, it is added a humidity sensor that produces events related to *humidity* topic. As this one did not exist previously, the ontology that stores all the topics is modified in order to incorporate it at the same abstraction level of *temperature* and *noise* ones. Thus, the *humidity* event is created and the *comfort* event is redefined. They will have the following structures:

- $\varepsilon_{humidity} = \{(\text{“percentage”}, float, x)\}$
- $\varepsilon_{comfort} = \{\varepsilon_{temperature}, \varepsilon_{noise}, \varepsilon_{humidity}\}$

As the *combination* entity will keep trying to make combinations between the information that is stored in the received events by using the *semantic* entity, it will now build a *comfort* event whose event nodes are extracted from humidity, temperature and noise events. Each time this entity builds a new event, it will notify it. Thus, the mobile application will receive these new *comfort* events automatically after adding the humidity sensor to the system and will be able to show humidity information to an end user without requiring any previous reconfiguration.

If the humidity sensor is removed from the scenario (figure 4, B), the comfort event will be redefined again so as to only contain information about the temperature and noise. Again, the *combination* entity will try to make combinations of the received events by using the *semantic* entity. Thus, it will now build *comfort* events by only using the information about temperature and noise. Therefore, it is possible to add or to remove participants to/from an ubiquitous system without requiring any changes in the subscriptions of any of the previously existing participants, as redefining events entails a dynamic reconfiguration of communications.

3. Technological Support

At the technological level, we have implemented a middleware based on the model and techniques proposed to show the feasibility of the approach. The middleware is defined as a software layer that is located between the operating system and the applications in order to hide the existing heterogeneity of different physical computer architectures, operating systems and programming languages, thereby, simplifying the process of transferring information between the different machines that are part of a distributed system [19].

The IceP communication protocol [20] has been used to codify network messages. An *event handler* is used by communication participants for sending and receiving events. Events are coded as *dictionary* structures. These dictionaries are collections of unique pairs of keys and values. Keys are, in this case, strings that represent the identifier of each event node (see Section 2.1) and values are arrays of bytes. These arrays of bytes represent values as specified by the IceP communication protocol.

The *communication abstraction layer* consists of a set of modules that hide which communication interface is used when transferring data from one machine to others. For example, by default, one module is provided for transferring data over a TCP/IP compatible interface to hide the inherent complexity of transferring messages over *sockets*.

Data distribution (*publish* function) may be centralized in a service or distributed between different communication participants, depending on the characteristics of the network. Thus, in multicast networks, participants will have support to distribute events to other participants without requiring any specialized entity, although communications would be unreliable (i.e., based on the UDP protocol). In order to support reliable communications (i.e., based on the TCP protocol), both *publication* and *subscription* entities, which are described in Section 2, have been implemented as services. In contrast with the unreliable distribution of events, in this case, communications are centralized and, thus, there are risks of bottlenecks in some communication scenarios.

The *semantic* entity has been developed as a service that makes use of a combination of an ontology described in OWL and Pellet reasoner [21].

Finally, in order to increase the portability of the middleware, its API is offered in several programming languages, like C++, Objective-C, Java, Python and PHP, and, in addition to Windows, Linux and MacOSX, it has been ported to iPhone and Android mobile platforms.

4. Related Work

Different works in the literature have dealt with the fulfilment of the requirements of ubiquitous systems by using the publish/subscribe communication paradigm. Specifically, some of them have introduced several notions of events and different publication and subscription techniques.

Courtenage [22] defines a way of specifying and detecting composite events in content-based publish/subscribe systems. Similarly, Data Distribution Service (DDS) [23] introduces the notion of MultiTopic in the description of a Topic, that is, a topic that results from the combination of several Topics. For doing that, it uses SQL-like expressions. In these proposals, event topics are not specified using an ontology that deals with the their conceptualization, they can not be modified dynamically and subscription and publishing techniques are not adapted to support some of the requirements associated with ubiquitous systems, such as providing a decentralized (or P2P) operation mode or supporting dynamic changes in the system (i.e., adding or removing applications, services or devices). By using an ontology, it will be supported to detect the consistency of the combinations, to formalize the information associated with each topic and to establish system-wide complex relations between topics.

Wang [24] and Petrovic [25] define two proposals that involve the usage of ontologies in order to support complex subscriptions, demonstrating the possibility of delivering efficient implementations of this kind of publish/subscribe systems. These papers are not focused on ubiquitous systems and, thus, they do not specifically address their requirements.

Cugola and Jacobsen [5] propose a publish/subscribe middleware for mobile systems that meets most of the requirements associated with ubiquitous systems. Our proposal is focused on how to deal with a dynamically changing environment in terms of adapting the information that is exchanged between the communicating participants, which is not the approach that follows the work referenced above.

5. Conclusions and Future Work

In this paper, we have proposed a model of redefinable events to support dynamic reconfiguration of communications in ubiquitous systems. We have also introduced some techniques for publishing and subscribing those events. The structure of the events and their intended semantics have been formally specified in an ontology, which enables automated reasoning based on Description Logics. Thus, our proposals give support to the mobility of communication participants between several networks, ensuring a valid data dissemination due to shared and formal semantics. Furthermore, this proposal has been implemented as part of a coordination middleware that supports both distributed and centralized publication techniques. Finally, this proposal may help in the design and development of Mobile Ad-Hoc Networks (MANETs), which consist of a set of mobile hosts that may exchange information mutually and roam around at their will without a base station to technologically support their communications [26] and which are gaining a special attention as a network topology to support communications in ubiquitous systems [27].

Also it is important to note that in the proposal, high level semantic information is interconnected with low-level layers, so as to provide functionality that was previously only offered at the application level. For example, currently, if an application needs to show information that may come from different sources and that only results from the combination of the information of those sources, this combination has to be "hard coded", affecting negatively to the flexibility of the application and its maintainability.

As future work we aim at extending the proposed middleware to support traditional coordination functionalities between communication participants (e.g., Linda coordination model [28]) in ubiquitous systems based on the publish/subscribe communication paradigm. Coordination capabilities will be enabled by appropriately combining both publishing and subscription techniques, formalized semantics of events and the *combination* service.

Finally, we plan to evaluate the performance of the middleware in several real scenarios and to compare the results with some existing middlewares for ubiquitous systems.

Acknowledgement

The Spanish Ministry of Education and Science funds this work through project TIN2008-05995/TSI.

References

[1] Weiser, M. "The computer for the 21st century". Scientific American, 265(3):94–104, September 1991.
[2] Ranganathan, A., Chetan, S., Campbell, R. "Mobile polymorphic applications in ubiquitous computing environments". In The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004., pages 402–411, Aug. 2004.
[3] Kiani, S. L., Riaz, M., Zhung, Y., Lee, S., Lee, Y.-K. "A distributed middleware solution for context awareness in ubiquitous systems". RTCSA '05: Proceedings of the 11th IEEE

International Conference on Embedded and Real-Time Computing Systems and Applications pp. 451-454. IEEE Computer Society, 2005.

[4] Baldoni, R., Cotenti, M., Virgillito, A. “The evolution of Publish/Subscribe communication paradigm”. Future directions in distributed computing: research and position papers, pp. 137-141, 2003.

[5] Cugola, G., Jacobsen, H.-A. “Using publish/subscribe middleware for mobile systems”. SIGMOBILE Mobile Computing Communications Rev., 6(4):25–33, 2002.

[6] Rumbaugh, J., Jacobson, I., Booch, G. “Unified Modeling Language Reference Manual, The (2nd Edition)”. Pearson Higher Education, 2004.

[7] Bates, P. C. “Debugging heterogeneous distributed systems using event-based models of behavior”. ACM Trans. Comput. Syst., 13(1):1–31, 1995.

[8] Friday, A., Roman, M., Becker, C., Al-Muhtadi, J. “Guidelines and open issues in systems support for ubicomp: reflections on ubisys 2003 and 2004”. Personal Ubiquitous Computing, 10(1):1–3, 2005.

[9] Kramer, J., Magee, J. “Dynamic configuration for distributed systems”. IEEE Transactions on Software Engineering, SE-11(4):424–436, April 1985.

[10] Shankar, C., Al-Muhtadi, J., Campbell, R., Mickunas, M. D. “Mobile gaia: A middleware for ad hoc pervasive computing”. IEEE Consumer Communications and Networking Conference (CCNC 2005), January 2005.

[11] Murphy, A., Picco, G., Roman, G. C. “Lime: A coordination model and middleware supporting mobility of hosts and agents”. Transactions on Software Engineering and Methodology (TOSEM), 15(3), Jul 2006.

[12] Levis, P., Culler, D. “Mate: A tiny virtual machine for sensor networks”. Proceedings of the international conference on architectural support of programming languages and operative systems, October 2002.

[13] Gruber, T. R. “A translation approach to portable ontology specifications”. Knowledge Acquisition, 5(2):199–220, 1993.

[14] Grau, B. C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U. “Owl 2: The next step for owl”. Web Semantics: Science, Services and Agents on the World Wide Web, 6(4):309 – 322, 2008. Semantic Web Challenge 2006/2007.

[15] Protégé ontology editor. Available online at: http://protege.stanford.edu/

[16] McGuinness, D. L., van Harmelen, F. “OWL Web Ontology Language”. W3C Recommendation, 10, 2004.

[17] Corsaro, A., Querzoni, L., Scipioni, S., Tucci, S., Virgillito, A. “Global Data Management”. Quality of Service in Publish/Subscribe Middleware, pages 1–19. IOS Press, 2006.

[18] Stankovic, J. A., Ramamritham, K. “What is predictability for real-time systems?”. Real-Time Systems, 2(4):247–254, November 1990.

[19] Bernstein, P. A. “Middleware: a model for distributed system services”. Communications of the ACM, 39(2):86–98, February 1996.

[20] Henning, M., Spruiell, M. “Distributed programming with ICE. Revision 3.3.1”. Available at: http://www.zeroc.com/doc/Ice-3.3.1/manual/toc.html.

[21] Pellet: OWL 2 Reasoner for Java. Available online at: http://clarkparsia.com/pellet/

[22] Courtenage, S. “Specifying and detecting composite events in content-based publish/subscribe systems”. Distributed Computing Systems Workshops, International Conference on, 0:602, 2002.
[23] Object Management Group (OMG). “Data Distribution Service for Real-Time Systems Version 1.2”. OMG Specification. January 2007.
[24] Wang, J., Jin, B., Li, J. “An ontology-based publish/subscribe system”. In Middleware ’04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, pages 232–253, New York, NY, USA, 2004. Springer-Verlag New York, Inc.
[25] Petrovic, M., Burcea, I., Jacobsen, H.-A. “S-ToPSS: semantic Toronto publish/subscribe system”. In VLDB ’2003: Proceedings of the 29th international conference on Very large data bases, pages 1101–1104. VLDB Endowment, 2003.
[26] Tseng, Y. C., Ni, S. Y., Chen, Y. S., Sheu, J. P. “The broadcast storm problem in a mobile ad hoc network”. Wireless Networks, 8(2/3), March 2002.
[27] Arabo, A., Shi, Q., Merabti, M. “Towards a context-aware identity management in mobile ad hoc networks (immanets)”. In WAINA ’09: Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops, pages 588–594, Washington, DC, USA, 2009. IEEE Computer Society.
[28] Carriero, N., Gelernter, D. “Linda in context”. Communications of ACM, 32(4):444–458, 1989.

Copyright Disclaimer