An XML-Based Model for Monitoring Pervasive Environments

Soraya Kouadri Mostéfaoui
University of Applied Sciences of Fribourg
Boulvard de Pérolles 80 - CP 32
Mobile Information Systems Laboratory
1705 Fribourg, Switzerland
kouadris@acm.org

Schahram Dustdar
Vienna University of Technology
Distributed Systems Group
Information Systems Institute A-1040
Wien, Argentinierstrasse 8/184-1, Austria
dustdar@infosys.tuwien.ac.at

Abstract

This paper presents a generic model for handling sensory information in pervasive environments. The used modelling concepts are founded on an XML based approach, in which sensory information is structured around a set XML schema files.

Keywords: sensory information, XML, XSD.

1. Introduction

In a typical context-based scenario; adding/removing sensors, or changing some parameters might be a difficult task. It may require rewriting major parts of the code, thus there is a need to hide all the hardware and software complexities; and to provide a simple way of interaction with the underlaying structure.

The model presented in this paper goes in this direction, it uses XML configuration files to allow to simply add/remove sensors, new information recipients, or to listen to other existing sensors [3]. Indeed, the different sensors are abstracted by internal software drivers to facilitate the interaction.

2. General Description

The data gatherer, the event handler and the communication module, are the core components of the model. Each one of these components has its own configuration and data pool; and can communicate with an arbitrary number of other components. In the following we briefly survey these components.

- The data gatherer gets the raw data from external sensors, and handles it. It does not work with the raw sensor data itself but simply transforms it. This component is also responsible for checking the persistence of the data. All the sensors are directly attached to this component. The sensors themselves can be either software or hardware; software sensors are sensors whose raw data are taken from a software means (e.g. a Web service, an aggregation of the data coming form other sensors or any another software component). The data gatherer has a further function; as it abstracts the sensors from the other modules, it has the possibility to create further virtual sensors that rely usually on aggregations. The data gatherer multiplexes the data and creates new sensors. The aggregation functions are clearly defined set of methods (e.g. SUM, AVG, etc.). The data gatherer has no real information about the meaning of the aggregated data and simply executes these methods.

- The event handler, handles all the information that comes from the data gatherer. It is the motor of the whole model, as it is the only component that is actually able to take decisions, whereas the other components simply handle configurations and transforms data into meaningful information. The decision making process however, is controlled by a configuration element which states which data fields to compare with which values. An event is only sent if the decision-making process recognizes it. The event data consists of the event and the data that have created that event. A data structure can actually react to several events (ie. temperature too low/too high), it can also react to several events at the same time. This could mean that several recipients want to react to different events.

  The continuous flow of data from the data gatherer, which is unaware whether an event has been sent or not, creates the effect that once an event has been triggered, it will be triggered again as soon as a new data arrives that satisfies the decision. This problem can be solved with a timeout value that defines how long to wait between two events. It makes sense to persist the

0-7803-9032-6/05/$20.00 ©2005 IEEE

Authorized licensed use limited to: Universitatsbibliothek der TU Wien. Downloaded on March 02,2010 at 04:52:03 EST from IEEE Xplore. Restrictions apply.

sent events, so that it is easier to ask when the last event has been sent.

- The last module of the model gets events and handles them according to its configuration. The handling process consists in communicating a message to a defined set of recipients over a number of push-channels. Each recipient may receive messages to several events and each event may trigger messages to several recipients. Instead of an event, it is also possible to send an information structure to a recipient (e.g. an XML file), where the recipient is not a human, but another machine, or simply another software module. In this way, the chain can be extended indefinitely. Messages can also be stored or buffered in a separate database for further handling.

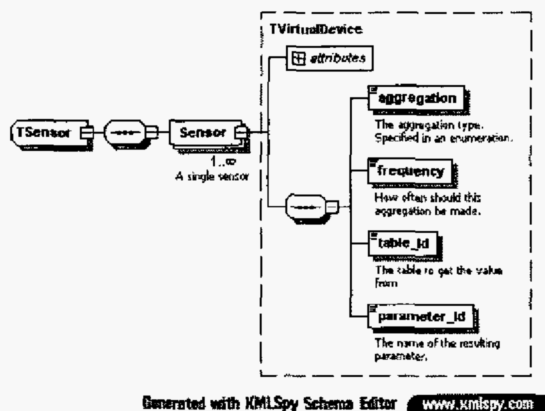

Figure 1. Sensor configuration schema

3. Model Configuration Using XSD Files

In this section we describe how the three modules are specified and configured using XSD files [1]. However due to the lack of space not all the XSDs files are presented, they can be found in [2] [3].

3.1 Data Gatherer Configuration

Several sensors (hardware or software) might be attached to the data gatherer module. This attachment follows a plug-and-play paradigm; the only requirement is that the data sent should be structured. It is worth emphasizing that: 1) the sensors communicate with the data gatherer over a network, this allows distributing the sensors over several machines; 2) the description of the structured data between the driver and the sensor plugger is the same for all sensors and very general; 3) complex sensors can be abstracted via vendor-specific drivers that simply have to conform to the communication interface (i.e. structure of the data to be sent); 4) a sensor configuration might be shared among several sensors. The major data structures that have to be defined in the data gatherer are shown below:

- sensor configuration: several types are possible and dependent of the sensor itself.
- structured data: this definition must be for every kind of data. The needed fields are: SensorId, Field with type, and value.
- data gatherer configuration: this configuration file specifies which data has to be stored where (a mapping between sensor information and data store). Furthermore it describes the aggregation functions and how they have to be relayed to the event handler.
- data store: the data store might for example be a relational data base (RDB) with a simple table which contains the following fields: Table, Column, Data type and Data.
- sensor data: the output of the data gatherer is a structure that is similar, if not equal, to the input data from the periphery sensors. It may, however, be virtual data, generated by an internal aggregation.

3.1.1 Sensor configuration

We describe now in details the structure of the sensor configuration file. The schema presented in Figure. 1, is used to describe several kinds of sensor information. In the general configuration data section, the schema defines where the sensor listens to (COM-Port), and where it sends the information. It is important to note, that a sensor can send data to more than one recipient.

3.1.2 Data gatherer configuration

The data gatherer needs a special configuration file which addresses the following questions:

- Where to listen to get data from the sensors?
- Where to send the data?
- How to aggregate which data?
- How to connect to the database?

As described in Figure. 2 the data gatherer configuration file contains the following elements:

- communication: special communication parameters like where to listen to and where to send the data.

Authorized licensed use limited to: Universitatsbibliothek der TU Wien. Downloaded on March 02,2010 at 04:52:03 EST from IEEE Xplore. Restrictions apply.

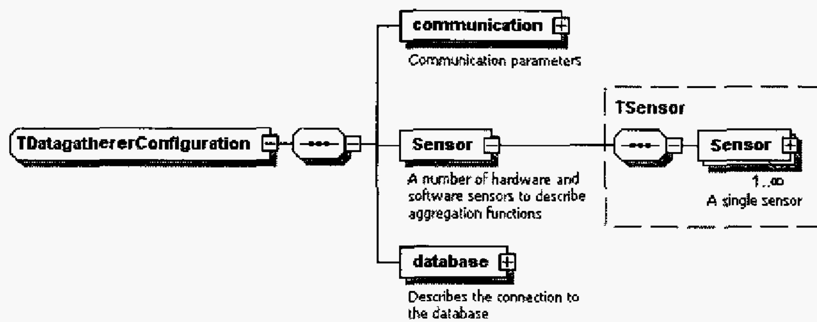

Figure 2. Date gatherer configuration schema

- virtualSensors: contains a list of virtualSensor that describes what and how to aggregate data and how the generated data is denoted.
- data base: this field describes how and where to connect to the context database.

3.1.3 Sensor communication

The data coming from a sensor to the data gatherer must be standardized. In this way, the data gatherer component does not have to care, where the data exactly comes from. Once the data from the sensor has arrived to the data gatherer, it is relayed to the event handler module. Therefore, no special structure is needed for the communication between the data gatherer and the event handler. The normal sensor communication schema will be used. It is worth emphasizing, that the existence of virtual sensors increases the communication, but the event handler does not remark anything about this, it simply gets data and handles it.

3.2 Event Handler Specification and Configuration

The event handler is the active motor of the model; it defines all the decisions, and sends them to the communication module for further handling. First the event handler receives structured data from the data gatherer module and must compare it to its own database of events; in case a value is within the definition of an event, this event is structured and sent to the communication module. The difficulty of this module lies mainly in the complex structure of the event definitions that have to allow all kinds of comparisons, as . Not every comparison is imaginable in the beginning; therefore modern software engineering methods like reflection are used to allow a high extensibility. The major data structures that have to be defined in the event handler are summarized below:

- event data: specifies the event that has been fired, the data associated with the event and the data associated with the boundary.
- configuration: it contains: the comparison definitions (Class, Operators), and event definitions (Id, sensor data to compare, comparison operation, value, timeout).

In the following we detail the main root elements of the event handler configuration schema, which can be found in [2].

- communication: contains all the data concerning the communication, it includes the ports to listen to, and where to send an event once it occurs.
- comparisons: contains all comparisons defined in the event handler module. Per comparison, a class is defined, so that introspection might be used to execute the comparison .
- events: it defines all events. An event is defined per sensor that might arrive from the data gatherer module. As the event handler does not make any distinction between real and virtual data, the structure is the same for all. The field comparison OperatorId must be defined in the comparisons list. Figure 3 shows the corresponding XSD file.
- database: contains all data necessary to access the database.

3.2.1 Event communication

This structure describes the content of the data when an event happens. It is also defined as an XSD. In the following code we illustrate with a simple example the communication between the event handler and the listening modules.

Authorized licensed use limited to: Universitatsbibliothek der TU Wien. Downloaded on March 02,2010 at 04:52:03 EST from IEEE Xplore. Restrictions apply.

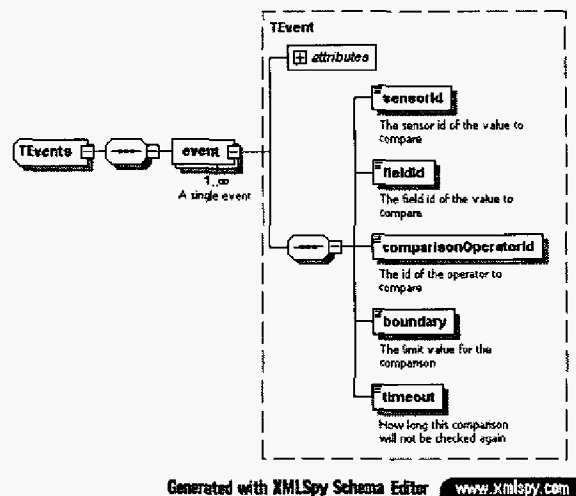

Figure 3. Event configuration schema

```
<event Id = "TEMPERATURE_TOO_HIGH">
    <parameter dataType = "double" Id = "ACTUAL_CELSIUS">
        <actual_value_double>38.20000</actual_value_double>
        <boundary_value_double>35</boundary_value_double>
    </parameter>
</event>
```

3.3 Communication Module Specification and Configuration

The communication configuration is splitted into three main configurations, the recipient configuration, message configuration and the transformation process configuration.

- recipient configuration: the list of recipients contains all recipients to which events will be sent. Furthermore it indicates which channels are chosen for communication.
- message configuration: together with the list of recipients goes the list of messages, which indicates the content of the data to be sent to the listener. Typically, a message listens only to one event, but it is possible that one message is used for several events. On the other hand, one event cannot have several messages assigned.
- transformation process configuration: the transformation process specifies the link between a channel and a message; as this tuple needs a transformation process to generate the final content, this matching is defined in a separate configuration structure.

More details on the XSD files corresponding to the recipient configuration, message configuration and transformation process configuration are available in [2].

4. Conclusions

In this paper we have introduced a model that can be used to attach an arbitrary number of heterogeneous sensors onto a wireless/wired network of distributed modules. Each module represents a specialist and communicates with an arbitrary number of other specialists. To describe this we are using the W3C schema language. This allows the use of well established standards, to guarantee the interoperability, extensibility and scalability of the model. Indeed new sensors can be easily plugged into the system without necessarily rewriting major parts of the code, powerful configuration files allow to simply add new information recipients or to listen to other modules. In order to have a validation of the model, implementation work has been conducted in the framework of a medical care scenario. Additionally two projects are under the way, emphasizing the use of the model in other scenarios [3]. One explores the use of the model in an in-house temperature control, and the other one in a fire alarm system.

References

[1] W3c xml schema. *www.w3.org/XML/Schema*, accessed June 2005.

[2] S. Kouadri. Context modelling xsd files. *http://diuf.unifr.ch/people/kouadris/CBSeC/XSD*, accessed May 2005.

[3] S. Kouadri. A sensory-oriented model for monitoring ubiquitous environments. *The 3rd International Workshop on Distributed and Mobile Collaboration (DMC'05) held in conjunction with The 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE'2005), Linkoping, Sweden*, June 2005.

Authorized licensed use limited to: Universitatsbibliothek der TU Wien. Downloaded on March 02,2010 at 04:52:03 EST from IEEE Xplore. Restrictions apply.