Dynamic Interaction Models for Web Enabled Wireless Sensor Networks

Maria Cecília Gomes, Hervé Paulino, Adérito Baptista, Filipe Araújo

CITI / Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal
{mcg,herve}@di.fct.unl.pt

Abstract—Wireless Sensor Networks (WSNs) are a fundamental technology for science in many domains, and their inclusion in Web environments, e.g. through Web services, allows for their open/standard access and integration. Although such Web enabled WSNs simplify data access, network parametrisation and aggregation, the available interaction models and runtime adaptation mechanisms are still scarce. Nevertheless, applications increasingly demand richer and more flexible interface accesses, for instance, the interaction model dynamic adaptation according to contextual information. To this extent, this paper discusses the relevance of the session and pattern abstraction on the design of a middleware prototype providing richer interaction models, as well as a few context-based dynamic adaptation mechanisms for Web enabled WSNs.

Keywords—Web enabled WSNs; Dynamic Interaction Models; Design Patterns;

I. INTRODUCTION

Simulation applications for unexpected, but extreme, events like large-scale flooding, hurricanes, severe droughts, etc., demand the access to different types of data collected across wide scale geographic areas, and for long periods of time. Only large amounts of diverse data support more precise information extraction and knowledge, concerning a better evaluation of such complex events.

Wireless Sensor Networks (WSNs) offer a good low cost solution for such large-scale environmental monitoring since they comprise a large number of sensor devices (either simple or more complex, static or mobile) deployed throughout the geographic area to be evaluated. WSNs allow hence the development of more or less elaborated applications [1] to which the interaction with the real world is a pressing requirement. Such includes not only those more traditional applications, but WSNs also allow the surge of novel applications, e.g. in the Participatory Sensing area [2] like urban traffic management or virtual communities’ support, relying on data acquisition and dissemination through mobile devices.

Nevertheless, one disadvantage of WSNs is still their low-level limited interfaces. High-level abstractions have been used to simplify their access, allowing representing WSNs, for instance, as data streams, databases, or through mobile agent models. Likewise, abstracting WSNs as Web services [3], [4] support their inclusion in Web environments, e.g. in the context of business processes. Namely, the service paradigm via standard Web technologies supports an uniform and simple access to WSNs, their parametrisation and aggregation, and the systematic access to collected data. A service-based access to WSNs also allows their integration with very different systems, since the service paradigm allows an uniform access to and aggregation of distinct entities. One example may be the seamless integration of WSNs for online/almost real-time data acquisition with Cloud-based applications consuming that data. In fact, and considering the perceivable trend on making everything accessible as a service (XaaS), the service concept may provide a powerful but simple abstraction for heterogeneous systems’ access, interaction, and integration, may those systems be Web enabled WSNs, Internet of Things entities [5], [6], mobile Clouds, Grid or Cloud computing services (for standardisation efforts in this area see [7]), etc.

Nevertheless, the access to those types of services may have requirements behind the traditional request/response interaction for which dynamic richer models are in need [8]. For instance, IoT entities having one single client may be interfaced by a stateful service, and Cloud computing services may interface stateful resources or long running activities, which need to be inspected in terms of resource consumption, dynamic requirements, or overall cost [9], [10]. In the case of Web enabled WSNs, sensors may have to be inspected and interrogated (e.g. in terms of sensor autonomy evaluation and sensing frequency), modified (e.g. sensor parametrisation), and sensing data may have to be acquired with different QoS depending on contextual information (e.g. sensing data streaming on an emergency situation versus periodic data notification for sensors’ autonomy preservation).

WSNs accesses may hence be modelled as web services interfacing stateful resources [11], [12] requiring the realization of a dynamic/variable state which has to be kept consistent along several message exchanges between a service and each one of its clients [13]. This is captured in the Web Services Resource Framework (WSRF) norm [14], which had its origin in the context of Grid computing [13], in order to represent the access to typical, long running, high Performance Computing applications. Consequently, such Web enabled WSNs may also benefit from richer/dynamic interaction models for sensor data acquisition and dissemination which, however, are not generally available in current...
solutions.
In the following sections, we describe the dimensions concerning such limitations and propose a solution towards richer interactions for Web enabled WSNs access. The subsequent sections describe the implementation architecture and an application scenario. The conclusions are described in the final section as well as future work.

II. Problem Dimensions

Consider an emergency application for a critical area prone to cyclic wild fire situations. Simulation applications in this context benefit from consuming almost real-time/online sensing data provided by different types of WSNs deployed in the area, e.g., temperature and humidity, in order to more accurately calculate a fire ignition probability [15]. Under normal conditions, temperature data acquisition from a single type of sensors may be enough. However, in the presence of drought weather conditions, more precise temperature data may be needed, e.g., collected from different sensors at different heights, and if a fire ignition does occur, different types of data like wind velocity and direction are also needed. In order to avoid the need for several independent client requests to acquire such data, one solution is to support their collective processing and dissemination as a single interaction action, similarly to what happens in the mashups concept. Moreover, the supporting system should allow the dynamic selection of those data sources at runtime.

Additionally, due to the low autonomy of typical sensing devices, the QoS in terms data acquisition rate and delivery should be low under normal situations, e.g., winter time for the fire application, and high in emergency situations. Moreover, assuming critical data is needed not only to the fire simulation’s execution, but also to firemen deployed in the area using mobile devices for their coordination, relevant data may now depend upon firemen geographical location (e.g., from the vicinities of their location). Likewise, if the mobile devices have already a low battery level, a data stream cannot be processed but sporadic data delivery is required instead. Whatever the client perspective, the most adequate solution is to provide flexibility on data sources’ dynamic selection and aggregation but also in terms of the data acquisition rate. Such may be captured in terms of the interaction model in use between the service and its clients, at some point in time, with the supporting system providing their dynamic modification based on context data. For instance, a Streaming model is preferable for a continuous sensing data delivery, a Producer/Consumer is necessary if there are data delivery requirements, and a Publish/subscribe model is more adequate whenever low rate data transmission is enough.

Furthermore, having defined a (more or less) complex monitoring scenario for such Web enabled WSNs accesses, it may be useful to reuse it, for example, in the context of a similar tornado simulation application for the area. Likewise, in case additional firemen corporations are deployed into the affected area, the contextual information perceived by the former firemen should be quickly and easily shared to the new ones.

Finally, considering that emergency protocols have to be precisely defined and known both by the actors in the field and authority entities, emergency support systems may provide some forms of pre-defined automatic dynamic reconfiguration capabilities concerning the evolution of a critical event. Such rules may be incorporated in those systems and be automatically triggered in face of particular events, e.g., sensing data values collected in a problematic area may trigger a switch from normal to an emergency situation.

Richer/dynamic interaction models are hence necessary on accessing Web enabled WSNs, and they should be captured in order to be shared among different clients and reused for similar situations. In the following, we propose a novel session-based abstraction to represent and contextualise such dynamic interaction models.

III. Proposed Solution

Our proposed solution is based on a) the Session concept to capture dynamic rich interactions with Web enabled WSNs, and on b) the Pattern concept to implement a confined, structured, and well defined mechanism for dynamic reconfiguration within a session context.

Sessions Capturing Dynamic Interaction Models: The Session concept represents the interaction context of a set of users accessing the same Web enabled WSN services, as well as the dynamic reconfiguration features possible within that context. A session includes:

1) The identification of the data sources plus the particular interaction model in use at some point in time for data dissemination. All client accesses within this session’s context obey the semantics of that interaction model, which
defines the service/users’ data and control flow dependencies. Basic interaction models are Client/Server, Publish/Subscribe, Streaming, and Producer/Consumer. Figure 1 depicts an example for a Wind data source, whose data is disseminated through a Streaming interaction model.

2) Management information, such as a unique session identifier used by new clients to join the session; the identifiers of the session’s current members; the identifier of the session’s owner, the sole that can perform explicit dynamic reconfigurations and terminate the session; and the session’s life time limit, which when expired causes the session’s termination and the consequent notification of all its members. If this time is unbounded, the termination must be explicitly requested by the owner. The session in Figure 1 has two clients and an unbound lifetime limit.

3) The possible adaptation mechanisms concerning structured and context-dependent adaptation in terms of WSNs service/users interactions. To this extent, the following two dimensions have to be considered: a) The interaction’s context which includes i) the context of the service client (e.g. a mobile device with limited autonomy or progressing to a different geographic area); ii) the interaction medium between the Web enabled WSN service and its user (e.g. the characteristics of the supporting communication networks); iii) the Web enabled WSN service’s context (e.g. data sources such as temperature or humidity sensor data). b) The interaction may evolve in time as a result of either an explicit user request or automatically triggered by the runtime system upon context change detection.

Pattern-based Dynamic Interaction Models: The pattern concept usage within a session’s context is twofold:

1) On one hand, patterns underlie an interaction model’s implementation in the context of a session. Such is accomplished following the ideas in [16], where pattern abstractions in the form of parametrisable Pattern Templates capture structure and behaviour with separation of concerns, allowing their flexible composition.

The implementation of a particular interaction model is based on the composition of one or more structural patterns with a behavioural pattern. On one hand, Structural Patterns capture a session’s “static view” in terms of the structural dependencies/relations among its members (e.g. a Façade or a pipeline) without specifying any restrictions in terms of data or control flows. The “dynamic view” is defined, on the other hand, by Behavioural Patterns like Producer/Consumer, Streaming, Publish/Subscribe, and so on. These characterise the dependencies in terms of data and control flows among a session’s members, as well as their role concerning the behavioural patterns’ semantics (e.g. roles of producer and consumer when considering the Producer/Consumer pattern). The left-hand side of Figure 2 presents the composition of a Façade structural pattern with the Publish/Subscribe behavioural pattern. The Façade captures the common interface for data dissemination to all clients in the session, and the behaviour defines how that data is disseminated to session’s clients.

Different interaction models enable the presentation of data flows with distinct quality services at different points in time. This allows diversity on accessing Web enabled WSN services/data sources, as well as for their modification when convenient. For example, the use of a Client/Server model to inspect a data source versus a Publish/Subscriber model to receive asynchronous event notifications.

The right-hand side of Figure 2, in turn, presents the implementation of an Aggregation model in the context of a session, which consists on the aggregation, and possible processing, of multiple data sources, and their dissemination. Such is supported by a hierarchical structure, namely a two-staged process (a two stage pipeline structure) consisting on an aggregation phase and a dissemination phase. Both phases must present the same behaviour, for instance, an aggregation of streams must be disseminated according to the Streaming behaviour. The logic used to combine the multiple data sources is parametrizable in the form of an aggregation function, which is parametrizable on the model’s definition. This approach accommodates the definition of application-specific stream processing techniques to filter the data, compute statistics, and so forth.

2) On the other hand, the pattern abstraction supports a structured dynamic adaptation mechanism which may be dependent on the current state of the interaction’s context. As a result each pattern can be directly reconfigured at runtime, both in the dimensions of structure and/or behaviour (e.g. to replace a behaviour by another one); Moreover, the adaptation/evolution of the system may be represented as a pre-defined sequence of patterns captured as a state machine (see Section IV).

IV. A Middleware for WSNs Adaptable Access

The proposed middleware implements the concepts described in the previous section to provide rich and dynamic
interaction models for Web enabled WSNs. It is implemented as Web accessible platform, upon which sessions can be shared by multiple geographically dispersed users.

The middleware’s architecture, depicted in Figure 3, follows a multi-tier model that cleanly separates the multiple concerns of the system, such as presentation, logic and data access. From a bottom-up perspective, the layers that compose the middleware are:

Data Acquisition - interacts with Web enabled WSNs, the data sources, providing a topic-based API. Upper layers can hence associate topics to data sources or define restrictions on those same sources. For example, a topic may refer to a stream of data produced by a given service or only to the items of the stream that obey a given condition (e.g. subscription of precipitation levels above 132 units, as depicted in the left-hand side of Figure 2).

Session Management - implements the session abstraction, supplying tools for session creation/termination; management, ranging from membership accounting to parameter configuration (e.g. lifetime specification) and; dynamic reconfiguration. Since a session may comprise geographically dispersed members, this layer exposes a simple Web service interface that is intended to be used by higher-level language APIs.

Session Centred High-Level API - provides a high-level session-centric interface for the cited capabilities. The remainder of this section will further detail the Session Management layer, the core of the middleware, and the Session Centred API used in the example of Section V.

A. Session Management Layer

A session hosts a single behaviour/interaction model to which all of its clients are automatically bound. This behaviour is defined at session creation time but may also change in time, as a response to a reconfiguration action. The client that creates a session is titled its owner and is the sole with permissions to perform reconfiguration actions that have a session-wide impact. The other members must comply to the session’s current configuration, and adapt to any consummated reconfiguration or leave. The composition of one or more structural patterns with a behavioural pattern provides the framework upon which sessions are implemented, as described in Section III.

Pattern-based Dynamic Reconfiguration: The reconfiguration mechanisms featured in the middleware have the purpose of adapting, in the context of a session, the way a particular client or a set of clients (the session’s members) interact with a set of stateful Web services. The separation of the session, structure, and behaviour concepts, and the way they are combined to support session execution, cleanly evidences the responsibility of each. For instance, the session contextualizes the overall interaction; a new client joining an existing session is captured as a structural reconfiguration independent from the behaviour (i.e. the new client has the same behaviour has the other existing clients in the same session); and the replacement of the session’s interaction model in use is captured as a behaviour reconfiguration independent from the structure (all clients in the session are notified of a new behaviour ruling data dissemination).

Additionally, the reconfiguration actions can be characterized as implicit (automatically triggered by the middleware) and explicit (requested by a client). Orthogonally, their scope may be confined to the tuning of the current interaction model, or have a session-wide impact, replacing the current model altogether. The conjunction of all the reconfigurations supported by the middleware defines a state machine whose description follows.

Explicit Reconfigurations: Valid reconfiguration requests may be issued by any member of any session, at any moment in time. Their purpose is twofold: to tune or to replace the current interaction model. Tuning requests are model dependent, and must conform to the currently active reconfiguration interface. For instance, setting the data rate is only available in the Streaming and Producer/Consumer models.

The remainder requests have a broader impact and thus have their semantics bound to the role of the client in the session. Only a reconfiguration request issued by the session’s owner may encompass the entire session. The other members are notified of such reconfiguration and will have to adapt to the new configuration or leave the session. Requests issued by some other member than the owner do not affect the target session. It is the client that is moved to another session fulfilling the required parameters. If no such session exists at the time, it is created on-the-fly.

Figure 4 illustrates the transitions of the state machine that are triggered by explicit requests. The ones that actually perform a state transition have been divided into three categories: Explicit - an explicit reconfigure request. Automatic - reconfiguration actions that, when in the scope of a Publish/Subscribe model, can be programatically associated to a particular topic subscription. As soon as the middle-

Figure 3. Overall architecture
ware receives a notification on that topic it automatically reconfigures the client, according to its role in the session (owner or regular member). **Add interaction** - addition of new data sources to the session. This reconfiguration forces the interaction model to become an aggregation, being that the dissemination model is inherited from the current configuration, e.g., adding a new source to a stream will result in the aggregation of two streams.

**Implicit Reconfigurations:** These constitute responses to changes in the context of the client, of the services, or of their communication channel. Their purpose is to ensure that the data flow between a session’s sources and clients is adjusted according to the session configuration parameters and the ability of the sources to meet these requirements.

Figure 5 presents the transitions of the state machine dedicated to this type of reconfigurations. Three scenarios are handled:

**Session out of reach** - this transition is triggered whenever the data source is no longer reachable. The session’s clients are notified of the incident and, from that point on, will only be able to interact with the source through the **Client**/**Server** model. Naturally, as long as the source is out of reach, any request will return an error message.

**No data** - when in the context of the **Stream** and **Producer/Consumer** interaction models, the absence of new data items causes the session to be reconfigured to **Publish/Subscribe**. Clients are notified of the data stream interruption, as well as of its resuming.

**Lower the rate** - the **Producer/Consumer** interaction model enables clients to consume data-streams at their own pace, which may be significantly slower or faster than their production rate. To support such feature, the middleware buffers data items on both ends of a client connection. In this context, the **Lower the rate** transition is triggered whenever the buffer that resides on the client end detects that it is no longer able to consume the data at the current pace. As the name implies, the reconfiguration lowers the rate to which the data items are sent to that particular client. Thereby, this reconfiguration targets a single connection, and not the whole session.

**B. The Java Session-Centred API**

A high-level session-centric API has been developed for the Java language. It exposes all of the middleware’s features, providing the means for applications to create, destroy, join and reconfigure existing sessions. Moreover, it specifies how an application can process incoming data items and react to consummated reconfigurations. Figure 6 showcases a simplified version of the API’s class diagram.

**Creating and Joining Sessions** - Session are instances of the **Session** class that can be parametrized with the topic(s) of the data sources, an interaction model (the default is **Client/Server**), a listener to handle incoming data (more on this ahead), and a duration in minutes (the default is **not limited**). All interaction models share a common interface (**InteractionModel**)) but provide specific reconfiguration interfaces (the methods of each class). The ability to join existing sessions is provided by the **join** method. It requires the identifier of the session to join and the listener to handle incoming data. The inquiry of which sessions and topics are currently active is possible through methods getActiveSession() and getAvailableTopics(), respectively.

**Reconfiguration Requests** - Three methods are provided for requesting explicit reconfigurations: reconfigureCurrent(), reconfigure() and addInteraction(). The first empowers the tuning of the current interaction model, while the remainder two instantiate, respectively, the explicit and add interaction transitions of Figure 4.

**Handling Incoming Data and Notifications** - All data received in the scope of a session must be processed by a
special handler that we refer to as listener. This handler must subtype abstract class NotificationListener and implement methods to process the reception of new application data items (processMessage()) and all possible exceptions and reconfiguration notifications (the remainder methods).

V. APPLICATION SCENARIO

As an application scenario, consider a fire detection application supporting a fire department responsible for a critical geographical area. The department is interested on receiving a notification whenever the temperature in the area raises to values above 50°C, and when this happens, a dynamic reconfiguration causes a switch from an alert state to a critical one contemplating the raise of the temperature above 80°C. Based on this last notification indicating a probable imminent fire scenario, the next step requires on-line (almost real time) data acquisition on wind-speed and direction, besides temperature. Such different data types should also be aggregated according to user defined criteria.

In case a secondary fire department is appointed to fight a fire in the same area, the application should provide them with access to the same data as the main fire department. Furthermore, if during the fire fighting period the main fire department decides to add another source of data, e.g. "Humidity", in order to gain a more precise information about the conditions in the terrain, this has to be acknowledged by the secondary fire corporation as well. Figure 7 represents such modifications in the context of a session capturing this application scenario.

The first image in Figure 7 (on the left-hand side) depicts a session created by the middleware to represent the interaction context between the session clients, namely the Main Fire Department (the session’s owner) and the Secondary Fire Department (the auxiliary corporation), and the available data sources accessible in the session, i.e. a Web enabled WSNs acquiring temperature data. The interaction model in use is the Publish/Subscribe, being the subscription topic: temperature values above 50, which defines an alert state.

If such temperature is observed, a user-defined dynamic reconfiguration takes place (First reconfiguration in Figure 7) where the subscription topic is modified. The fire departments are now interested in being notified when the temperature rises above 80°C, which, if measured, indicates a critical situation. Note that interaction model is left unaltered, and thus both departments are notified of this event.

On such scenario, a second automatic dynamic reconfiguration (Second reconfiguration) is triggered to build an aggregation of multiple data sources. In the face of a critical situation, temperature data inspection is not enough, and new data sources on wind speed and direction are dynamically added to the session context. Data collected from different types of Web enabled WSNs may hence be aggregated in the context of the session and processed according to a user-defined aggregation function. Moreover, for a precise evaluation of the fire situation (e.g. if a fire ignition is imminent or has already occurred), a continuous data flow from the sensor devices monitoring the area is now mandatory. Such is also depicted in the new configuration, where the interaction model used for both the aggregation and dissemination stages is the Streaming model.

Finally, the case when additional data sources are still needed, e.g. on humidity values, illustrated by the Third reconfiguration. The aggregation model remains as the underlying interaction model, but a new Web service interfacing WSNs has been added, allowing the definition of a different aggregation function for processing all the types of incoming data.

Note that this session’s context captures the subordination of the Secondary Fire corporation in the field to the Main Fire Department in terms of relevant collected data and the associated response. For instance, Listing 1 sketches the creation of a session with a Publish/Subscribe interaction model used to notify temperature values. When those values exceed a minimum threshold, the above critical situation is established and a pre-defined dynamic modification takes place (“Second reconfiguration” in Figure 7). This reconfiguration is defined by the Session owner (Main Fire Department) and consists on an Aggregation of streams on temperature, wind direction, and wind speed values, as depicted in Listing 1.

```
Listing 1. Main Fire Department’s session

Aggregation agg = new Aggregation("Temperature", pb.
      .getInteractingModel().getStreams().getAggregationFunction());
PubSub pb = new PubSub();
pb.onNotification("Temperature", "WindSpeed", "WindDir").
    agg.
      new TemperatureWindSpeedWindDirectionListener());
Session s = new Session("Temperature", pb,
    new TemperatureListener());
s.start();
```

Listing 2. A new fire corporation joins the session

```
public class TeamListener extends AbstractListener {
  public void processMessage(DataItem msg) {
    gui.display(msg.getContents());
  }
  public void reconfiguration(ReconfNotification n) {
    gui.displayNotification("Temperature value reached dangerous levels – Streaming mode activated.");
    getTSession().setListener(new DangerLevelListener());
  }
}
```

Listing 3. Sketching the implementation of TeamListener

In order to acknowledge and handle the events occurring in the context of the session, the above Secondary Fire
corporation (or other novel clients joining the session at some point in time) has to implement the TeamListener handler as it is sketched in Listing 3. The disclosed methods handle the reception of data items, displaying them in a user interface (gui), and defines a new listener able to process the reconfigurations possible in a new session’s state.

VI. RELATED WORK

The work in [17] presents one solution for self-adaptability of service-generated data streams targeting problems such as data loss or delays associated with communication networks disruptions. However, interaction models are not present as explicit configuration options considering service interactions as proposed here. Although the cited approach does implement a (sophisticated) Producer/Consumer interaction model, such is restricted to the support system (i.e. it is not explicitly visible at the point-to-point interaction level between a service and its user). Furthermore, in [17] there is no reference on the possibility of dynamically adding new data flow consumers or additional data sources, as we proposed in the session’s context.

Some other works use the pattern concept for systems’ self-adaptation mechanisms but in a different way, namely, by making use of reconfigurable Architectural Patterns for system definition [18]. The architecture of the Publish/Subscriber pattern, for instance, allows the reconfiguration of publishers, subscribers, and subscribed events. The Master/Slave pattern also allows the addition of new slaves to optimise task execution [19]; such is also incorporated in our solution within the context of a session. In spite of such reconfigurable system architecture definition, these works do not provide a session capturing an interaction’s context, nor a pattern-based system evolution based on pre-defined rules conform to those pattern’s semantics. Finally, our solution is based on the work by [16] which, however, does not target the stateful services’ domain, does not provide a session context, nor implements a state machine for pattern-based system evolution.

Regarding middlewares for Web enabled WSNs, existing approaches do not focus, to the best of our knowledge, on the dynamic reconfiguration of the client-WSN interaction model. Among these we highlight: 1) 52° North [12], the most known implementation of the Sensor Web Enablement (SWE) [3], a set of models and Web service interfaces proposed by the Open Geospatial Consortium for the Web integration of sensor systems. The models focus the description of sensor systems and their capabilities to collect and process observations, while the services address the collection, storing and dissemination of sensor reading and alerts (notifications). 2) Global Sensor Network (GSN) [20], which aims at building a sensor Internet by connecting virtual sensors, abstracting data-streams originating from either a WSN or from another virtual sensor; SQL queries can be performed on top of these virtual sensors. 3) SenSer [4], a generic middleware for the remote access and management of WSNs, being the latter virtualised as Web services, in a way that is programming language and WSN development platform independent. Among its distinguishable properties are the ability to filter the acquired data and to submit WSN reprogramming requests.

VII. CONCLUSIONS AND FUTURE WORK

This work proposes a solution based on Session concept to capture, contextualise, and reuse, richer dynamic interaction models to Web enabled WSNs, allowing their use across different application scenarios. A session embodies the common interaction characteristics relating a set of users accessing the same service, and all clients perceive the same events occurring in the session’s context. A Session also contextualises the possible dynamic adaptations both in terms of the WSN’s service, communication medium, or clients’ contexts. Sensing data, the transfer rate, or a client’s mobile device autonomy, may all trigger the modification of the interaction model. Both the interaction models, and the rules for their dynamic adaptation, rely on in turn on the pattern concept and depend on individual pattern semantics. Additionally, a state machine defines the system’s evolution based on pre-defined pattern-based rules. Such per-pattern reconfigurations, being well-defined, allow dynamic reconfiguration automation and contribute to limiting, to some extent, the impact of system’s dynamic reconfiguration. The performance evaluation in terms of the
overhead of one additional middleware layer between a Web enabled WSN (Senser platform) and its users is one point that unfortunately is missing in this paper but which will be studied in the near future. Likewise, more application scenarios are needed in order to evaluate the expressiveness of the model. Nevertheless, it is our opinion that the novel work described in this proposal concerning richer dynamic interaction models for Web enabled WSNs, opens several interesting further developments concerning this context. For instance, the aggregation of session-based interactions in the form of workflow dependencies; and the deployment of the developed middleware in a Cloud computing platform for ubiquitous and reliable access. These cases are already under development.

ACKNOWLEDGMENT

This work was partially funded by the PEst-OE/EE/UI0527/2011 Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) - 2011-2012. The collaborations of Prof. Maria Cecília Gomes and Prof. Hervé Paulino are partially supported by projects SITAN–PTDC/EEI/EIA/113729/2009 and MACAW–PTDC/EEI-EIA/115730/2009, respectively.

REFERENCES


