Accessing Wireless Sensor Networks Via Dynamically Reconfigurable Interaction Models

Maria Cecília Gomes, Hervé Paulino, Adérito Baptista, and Filipe Araújo,
CITI/Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Caparica, Portugal

***Abstract* — The Wireless Sensor Networks (WSNs) technology is already perceived as fundamental for science across many domains, since it provides a low cost solution for environment monitoring. WSNs representation via the service concept and its inclusion in Web environments, e.g. through Web services, supports particularly their open/standard access and integration. Although such Web enabled WSNs simplify data access, network parameterization and aggregation, the existing interaction models and run-time adaptation mechanisms available to clients are still scarce.**

Nevertheless, applications increasingly demand richer and more flexible accesses besides the traditional client/server. For instance, applications may require a streaming model in order to avoid sequential data requests, or the asynchronous notification of subscribed data through the publish/subscriber. Moreover, the possibility to automatically switch between such models at runtime allows applications to define flexible context-based data acquisition. To this extent, this paper discusses the relevance of the session and pattern abstractions on the design of a middleware prototype providing richer and dynamically reconfigurable interaction models to Web enabled WSNs.

***Keyword* — *Web Enabled* Wireless Sensor Networks, 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 complex events of this kind.

Wireless Sensor Networks (*WSNs*), in particular, offer a good low cost solution for such large-scale environmental monitoring since they comprise a high number of sensor devices deployed throughout the geographic area to be evaluated. Current WSNs may include different types of sensors, spanning from simple and static devices to increasingly complex mobile devices. 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 more traditional applications like the ones mentioned above, but WSNs also allow the surge of novel ones. This is the case of the *Participatory Sensing* area [2] where applications like urban traffic management or virtual communities' support typically rely on data acquisition and dissemination through mobile devices (e.g. using sensors embedded in private cars and mobile phones).

Nevertheless, one disadvantage of WSNs is still their low-level limited interfaces. To this concern, high-level abstractions have been used to simplify WSNs access, allowing their representation as data streams, databases, or through mobile agent models, for instance. Likewise, abstracting WSNs as Web services [3][4] allows their inclusion in Web environments, e.g. in the context of business processes. Namely, the service paradigm via standard Web technologies supports a uniform and simple access to WSNs, their parameterization 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 provides a uniform access to, and aggregation of, distinct entities. One example may be the seamless integration of WSNs providing 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 (*IoT*) [5][6], *Grid* or *Cloud* computing services (for standardization efforts in this area see [7]), etc.

Nevertheless, the access to those types of services may have requirements behind the traditional request/response interaction, demanding therefore dynamic/richer interaction models [8]. For instance IoT entities having one single client (the owner) may be interfaced through a stateful Web service. Cloud computing services, in turn, 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]. Considering specifically Web enabled WSNs, sensors may have to be inspected/interrogated (e.g. in terms of sensor autonomy evaluation and sensing frequency) and also be modified (e.g. sensor parameterization).

DOI: 10.9781/ijimai.2012.176

Additionally, 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 consequently be modeled 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 benefit from richer/dynamic interaction models for sensor data acquisition and dissemination that however are not generally available in current solutions.

The following sections describe the dimensions concerning such limitations and propose a solution towards richer interactions for Web enabled WSNs access. Subsequent Sections IV and V describe, respectively, 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. In order to more accurately calculate a fire ignition probability [15], simulation applications in this domain benefit from consuming almost real-time/online sensing data provided by different types of WSNs deployed in the area, e.g. temperature, humidity and wind characteristics' monitoring. Under normal conditions, temperature data acquisition from a single type of sensors may be enough. However, in the presence of draught weather conditions, more precise temperature data may be needed, e.g. collected from different sensors at different heights. Moreover, if a fire ignition does occur, different types of data like wind velocity and direction are also needed.

In case a client uses a traditional request/reply interaction model to collected sensing data, several independent client requests are necessary in order to process enough quantities of different data. One solution is to support the collective data 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, and 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.

Other requirements may also be considered. For instance, critical data may be needed not only to the fire simulation's execution, but also to the firemen deployed in the area. Namely, these may be using mobile devices for their coordination and relevant data may now depend upon their geographical location (e.g. data collected at the vicinities of the firemen's position). Likewise, if the mobile devices have already a low battery level, a data stream cannot be processed anymore, but sporadic data delivery is required instead.

Whatever the clients' perspective, the most adequate solution is to provide flexibility on data sources' dynamic selection and aggregation, and also in terms of the data acquisition rate. Such may be supported through selecting an adequate interaction model between the service and its clients, at some point in time. The supporting system should also provide 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/subscriber model is more adequate whenever low rate data transmission is enough.

Having defined such a (more or less) complex monitoring scenario for different Web enabled WSNs data sensing, its reuse for related clients/applications may also be useful. For example, the described scenario could be used in the context of a similar tornado simulation application for the area. Likewise, in case additional firemen corporations are deployed into the affected location, the contextual information perceived by the former firemen should be quickly and easily shared to the new ones.

Contextual information sharing may also support the coordination of relevant agents, e.g. considering that emergency protocols have to be precisely defined and known both by the actors in the field and authority entities. Based on a common context, emergency support systems may hence enforce 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.

Therefore, it is our opinion that richer/dynamic interaction models are necessary on accessing Web enabled WSNs and that they should be captured allowing their sharing among different clients and reuse for similar situations. In the following, we propose a novel session-based abstraction to represent and contextualize such dynamic interaction models.

III. Proposed Solution

The conceptual view of the proposed solution is depicted in Fig. 1. The middleware layer hides the details inherent to accessing Web enabled WSNs and provides an interaction context to clients, either individual or to a set (e.g. clients which may benefit from sharing a particular interaction). The 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.

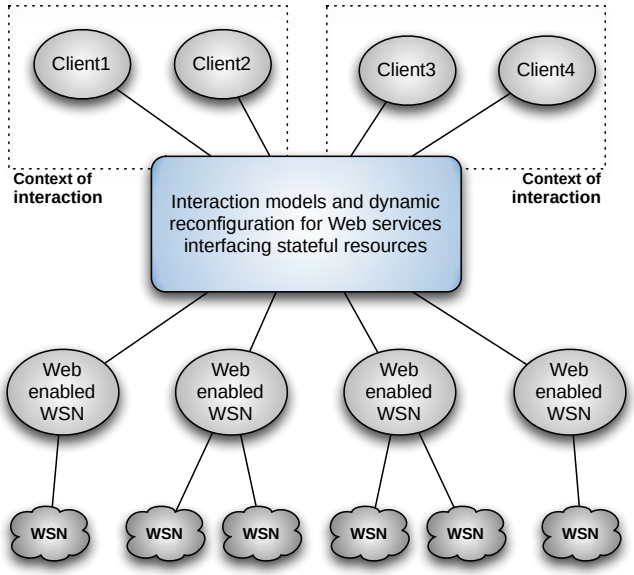

Fig. 1. Conceptual view.

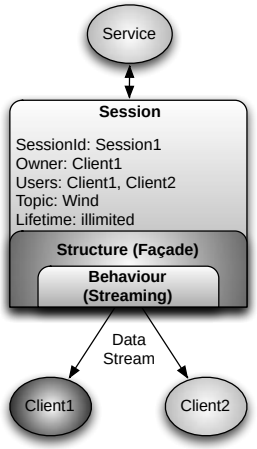

Fig. 2. Session abstraction with a Stream-based interaction model.

A. *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/Subscriber, Streaming, and Producer/Consumer. Fig. 2 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 owner must explicitly request the termination. The session in Fig. 2 has two clients and an unbound lifetime limit.
3) The possible adaptation mechanisms consisting of structured and context-based dynamic reconfigurations. These depend on the characteristics of the WSNs service/users interaction context and may also be pre-defined:
   - *The interaction's* context 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. services representing relevant data sources like temperature or humidity sensing data whose critical values have to be acknowledged).
   - *System evolution results from on-demand/pre-defined interaction models' dynamic modifications*. Users may explicitly require dynamic reconfigurations, or these may be automatically triggered by the runtime system based on pre-defined rules and upon change detection of the cited interaction context.

B. Pattern-based Dynamic Interaction Models

Within a session's context, the *pattern concept* is used both

- To implement the interaction model in use by all clients belonging to the session at some point in time; and
- To provide a structured dynamic adaptation mechanism ruling a session's evolution.

Implementing the Session's Interaction Model

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 parameterized *Pattern Templates* capture structure and behavior 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 behavioral pattern. *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 *Behavioral Patterns* like *Producer/Consumer*, *Streaming*, *Publish/Subscriber*, and so on. These characterize the dependencies in terms of data and control flows among a session's members, as well as their role concerning the behavioral patterns' semantics (e.g. roles of *producer* and *consumer* when considering the Producer/Consumer pattern).

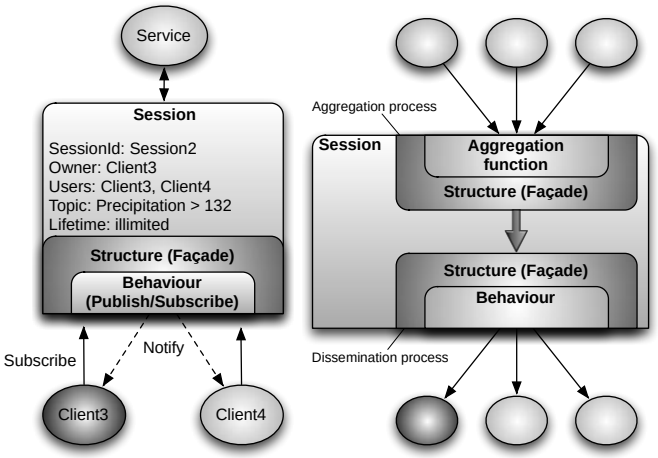

Fig. 3. Session's Interaction Models implemented as the composition of structural and behavioral patterns.

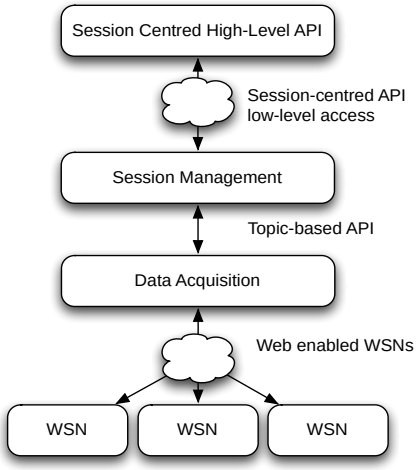

Fig. 4. Overall architecture.

The left-hand side of Fig. 3 (a) presents the composition of a Façade structural pattern with the Publish/Subscriber behavioral pattern. The Façade captures the common interface for data dissemination to all clients in the session, and the behavior 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 Fig. 3 (b), 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 of an aggregation phase and a dissemination phase. Both phases must present the same behavior, for instance, an aggregation of streams must be disseminated according to the Streaming behavior. The logic used to combine the multiple data sources is defined in the form of an *aggregation function* parameterized upon the model's definition. This approach accommodates the definition of application-specific stream processing techniques to filter the data, compute statistics, and so forth.

Structured Dynamic Adaptation

The pattern abstraction also supports a structured dynamic adaptation mechanism dependent on the current state of the interaction's context. As a result

a) Each pattern can be directly reconfigured at runtime, both in the dimensions of structure and/or behavior (e.g. to replace a behavior by another one);
b) 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 providing rich and dynamic interaction models for Web enabled WSNs. It is implemented as a Web accessible platform upon which sessions can be shared by multiple geographically dispersed users.

The middleware's architecture, depicted in Fig. 4, 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 Fig. 3 (b)).
- ***Session Management*:** implements the session abstraction, supplying tools for session creation/termination; session management, ranging from membership accounting to parameter configuration (e.g. lifetime specification); and possible dynamic reconfiguration mechanisms. Since a session may comprise geographically dispersed members, this layer exposes a simple Web service interface intended to be used by higher-level language APIs.
- ***Session-Centered 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-Centered API* used in the example of Section V.

A. Session Management Layer

A session hosts a single behavior/interaction model to which all of its clients are automatically bound. This behavior must be defined when the session is created 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

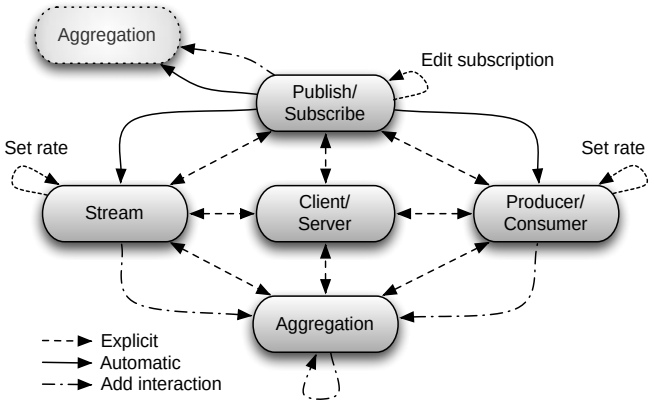

Fig. 5. Reconfiguration state machine: explicit reconfigurations.

permissions to perform reconfiguration actions that have a session-wide impact. The other members must comply with the session's current configuration, and adapt to any consummated reconfiguration or leave. The composition of one or more structural patterns with a behavioral pattern provides the framework upon which sessions are implemented, as described in Section III.B.

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) interacts with a set of Web services.

The separation of the *session*, *structure*, and *behavior* concepts, and the way they are combined to support session execution, cleanly evidences the responsibility of each one. For instance, the session contextualizes the overall interaction; a new client joining an existing session is captured as a structural reconfiguration independent from the behavior (i.e. the new client has the same behavior as the other existing clients in the same session); the replacement of the session's interaction model in use is captured as a behavior reconfiguration independent from the structure (all clients in the session are notified of a new behavior 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.

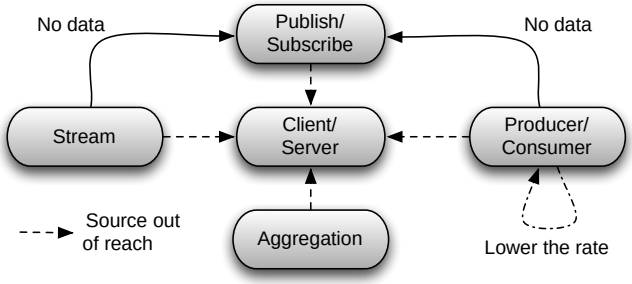

Fig. 6. Reconfiguration state machine: implicit reconfigurations.

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.

Fig. 5 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 programmatically associated to a particular topic subscription. As soon as the middleware 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, the service, or 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.

Fig. 6 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 they will only 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

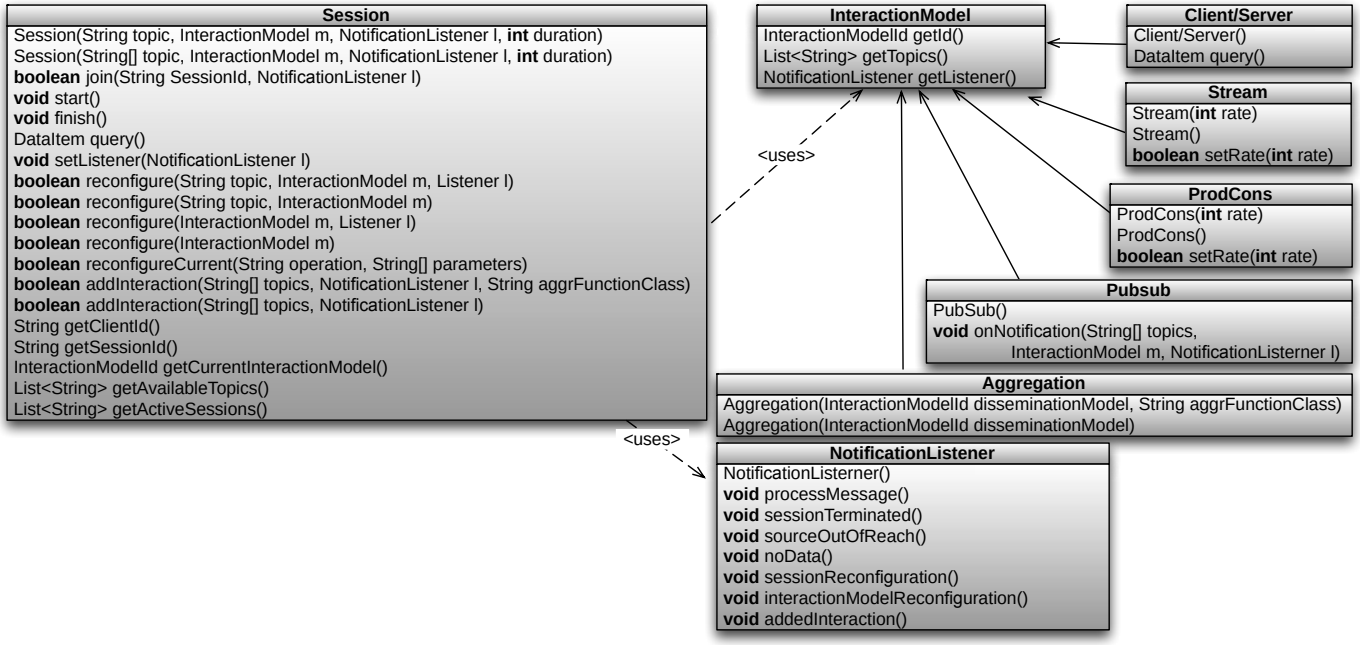

Fig. 7. API's simplified class diagram.

new data items causes the session to be reconfigured to Publish/Subscribe. Clients are notified of both the data stream's interruption and 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-Centered 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. Fig. 7 showcases a simplified version of the API's class diagram.

Creating and Joining Sessions

Session are instances of the Session class that can be parameterized 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 *unbound*). 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 be joined 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 the *explicit* and *add interaction* transitions of Fig. 5, respectively.

Handling Incoming Data and Notifications

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

V. Application Scenario

The application scenario chosen to illustrate some capabilities of our proposal belongs to the domain of the *Data Driven Applications and Systems* (DDDAS) [21]. These applications are characterized by the need to dynamically incorporate sensing data into a running simulation. Inversely, the simulation should also be able to dynamically parameterize how such sensing data is collected (e.g. restricting data acquisition to the most affected areas in order to reduce data processing). Our example describes only a partial scenario in the context of a fire monitoring and simulation application, as introduced in section II. Namely, a session contextualizes the dynamic aggregation of sensing data collected in a critical area, and typical clients to this session are fire workers and a fire evolution simulation. These clients may hence share the

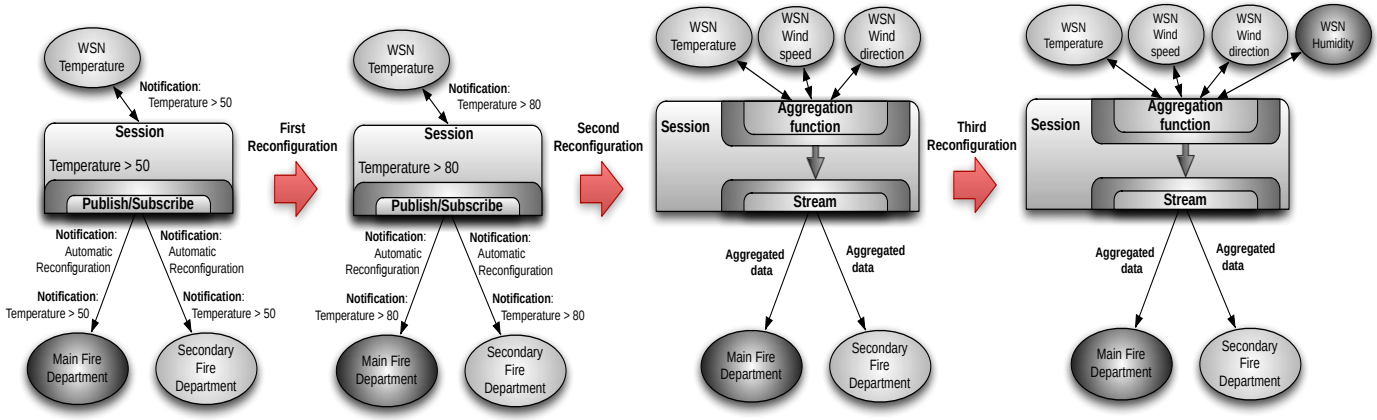

Fig. 8. Dynamic reconfigurations in a session context.

same context both in terms of collected data and the used interaction model for that data dissemination; additionally, all clients in the session are notified of the same dynamic reconfiguration events. The example in the next sub-section describes only the perspective of the fire workers.

A. Wildfires notification application

Consider a fire detection application supporting a fire department responsible for a critical geographical area prone to recurrent wild fire events. The department is interested on receiving a notification whenever the temperature in the area rises to values above 50º Celsius. Furthermore, when this happens, a dynamic reconfiguration should cause a switch from an *alert state* scenario to a *critical state* contemplating the raise of the temperature above 80º. Based on this last notification indicating a probable imminent fire ignition, 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's 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 more precise information about the conditions in the terrain, this has to be acknowledged by the secondary fire corporation as well. Fig. 8 represents such modifications in the context of a session capturing this application scenario.

B. System dynamic evolution

The first image in Fig. 8 (on the left-hand side) depicts a session created by the middleware including:

1) The interaction context between the session clients, namely the *Main Fire Department* (the session's owner) and the *Secondary Fire Department* (the auxiliary corporation).
2) The available data sources accessible in the session, i.e. a Web enabled WSNs acquiring temperature data.
3) The interaction model in use is the *Publish/Subscribe* being the subscription topic: temperature values above 50, which defines an alert state.
4) The dynamic reconfiguration rules.

Namely, if such temperature value of 50 is observed, a user-defined dynamic reconfiguration takes place (*First reconfiguration* in 8) modifying the subscription topic. The fire departments are now interested in being notified when the temperature reaches 80º or above which indicates a critical situation. Note that the interaction model is left unaltered, and thus both departments are notified of this event.

On such scenario, another 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, is 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.

```
// Listeners
Listener ownerListerner1 = new TemperatureListener50();
Listener ownerListerner2 = new TemperatureListener80();
Listener ownerListerner3 = new
TemperatureWindSpeedWindDirectionListener();
// Interaction models
Aggregation fire =
            new Aggregation(InteractionModel.STREAM,
                        "fire.HPAggregationFunction");
PubSub critical = new PubSub();
critical.onNotification({"Temperature", "WindSpeed",
                        "WindDir"}, fire, ownerListerner3);
PubSub alert = new PubSub();
alert.onNotification( "Temperature>80", critical,
                        ownerListerner2);
// Session creation
Session s = new Session("Temperature", alert,
                        ownerListerner1);
s.start();
…
// Later, during the fire fighting
Listener ownerListerner4 = new HumidityListener();
s.addInteraction("Humidity", ownerListerner4);
```

Listing 1: Main Fire Department's session

Note that this session's context captures the subordination, in the field, of the *Secondary Fire Corporation* 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 Fig. 8). 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.

To share the same session context and subsequently be notified of the same events as the owner - including dynamic reconfigurations - the *Secondary Fire Corporation* has to know this session's identifier and join it as specified in Listing 2.

```
Session s = Session.join(sessionId, new ClientListener1());
```

Listing 2: A new fire corporation joins the session

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 ClientListener1 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 define a new listener able to process the reconfigurations possible in a new session's state.

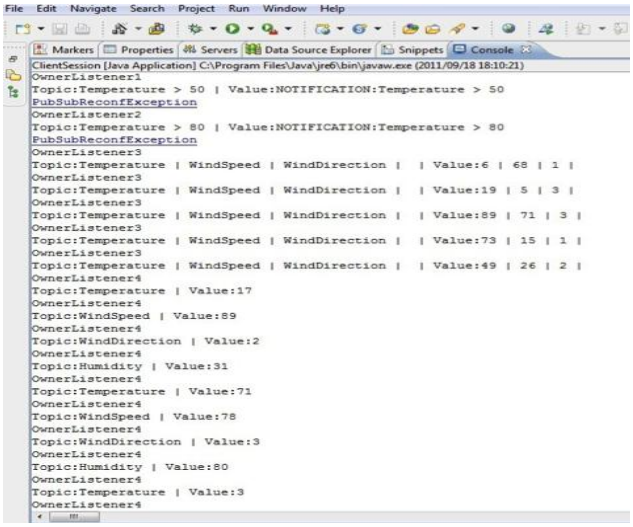

Fig. 9. Output of the main fire department - owner.

```
public class ClientListener1 extends AbstractListener {

  public void processMessage(DataItem msg) {
    gui.display(msg.getTopic() ,msg.getContents());
  }
  public void InteractionModelReconfiguration(
                    ReconfException n) {
    gui.displayNotification(n.getTopic(), n.getReason());
    getSession().setListener(new ClientListerner2());
  }
}
```

Listing 3: Sketching the implementation of ClientListener1

C. Example output

In the context of the previous particular scenario Figs. 9 and 10 illustrate the reception, on both fire departments, of the data values and notifications disseminated in the context of the session. The display of data values complies with the following format:

Topic*: subscribed_topic* | Value*: received_value*

while the display of notifications adheres to format:

Topic*: subscribed_topic* | Value*:*NOTIFICATION:*reason*

Reconfiguration notifications are disseminated to all members of a session, including its owner, which pre-defined the reconfiguration request. This approach entails a uniform way to react to a given notification, regardless of the member's role in the session.

Fig. 10 also shows the situation when the Main Fire Department requests a novel stream on humidity values (third reconfiguration on Figure 7) but an aggregation function is not supplied in the invocation of addInteraction() (last line of Listing 1). As a consequence, the messages are no longer

aggregated by the middleware, who simply forwards them. Such behavior can be observer on both figures from the point the fourth listener takes action.

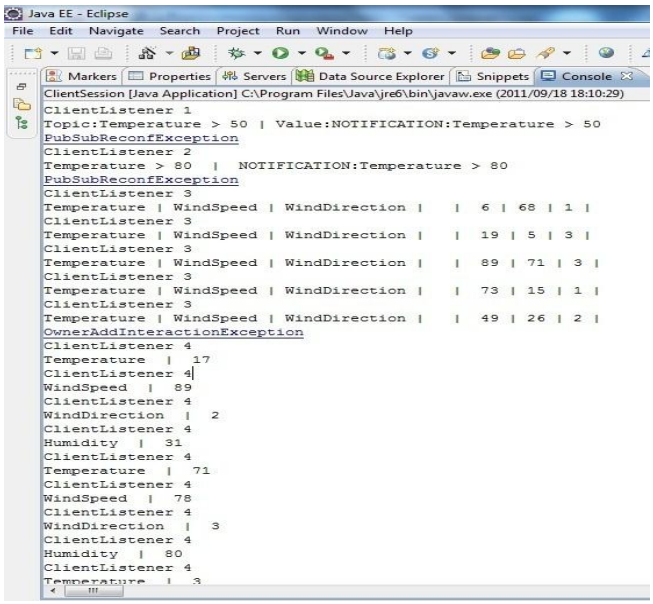

Fig. 10. Output of the second fire department – participative user.

VI. Related Work

To the best of our knowledge, existing middleware platforms for Web enabled WSNs do not address client-WSN interaction model's dynamic reconfiguration concerns nor provide a session abstraction to capture and reuse such dynamicity. Among those platforms we highlight:

52º 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 on 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).

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.

SenSer [4], a generic middleware for the remote access and management of WSNs, being the latter virtualized as Web services, in a way that is programming language and WSN development platform independent. Its distinguishable properties include the ability to filter the acquired data and to submit WSN reprogramming requests.

As for the presence of the pattern concept on system's dynamic adaptations, 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 described in our proposal. 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 to 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 reconfigurable *Architectural Patterns* for adaptable system's 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 optimize 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 provide a session abstraction to contextualize and reuse dynamic interaction models, nor implements a state machine for pattern-based system evolution.

VII. Conclusions and Future Work

Current applications relying on WSNs for large-scale environment monitoring require adequate abstractions for network access and parameterization, and sensing data acquisition. However, applications also increasingly request the seamless integration of WSNs in heterogeneous and dynamic complex systems, what is possible via the service concept. Moreover, the access to sensing data requires richer interaction models besides the traditional synchronous request/reply model, for example the *Publish/Subscribe* and *Streaming* models. Based on such Web enabled WSNs this work proposes a *session* abstraction in order to capture, contextualize, and reuse diverse richer dynamic interaction models to those services.

A session embodies the common interaction characteristics relating a set of users accessing the same service at some point in time, and all perceive the same events occurring meanwhile in the session's context. A session also contextualizes the possible dynamic adaptations both in terms of the service, the communication medium, or the clients' contexts. For instance the sensing data, the data transfer rate, or a client's mobile device autonomy, may all trigger the modification of the interaction model. Furthermore, both the interaction models and the rules for their dynamic adaptation rely on the *pattern* concept and depend on individual pattern semantics. The system's evolution is captured in a state machine based on pre-defined pattern-based rules. Being well defined, such per-pattern reconfigurations allow adaptation automation and contribute to limiting, to some extent, the impact of the dynamic reconfiguration upon the overall system.

The performance evaluation in terms of the overhead of one additional middleware layer between a Web enabled WSN and its users (SenSer platform [4]) 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 such novel session-based abstraction opens several interesting further developments concerning the inclusion and aggregation of diverse WSNs sensing data in different domains. For instance the aggregation of session-based interactions may be captured in the form of workflow dependencies and be used in ambient intelligence contexts and participatory sensing applications. Furthermore, the proposed middleware's deployment in a Cloud computing platform may provide clients a ubiquitous and reliable access to sessions. These cases are already under development.

ACKNOWLEDGMENT

The authors would like to thank Professors Omer Rana and José Cardoso e Cunha for the initial ideas on pattern and service abstractions, which ultimately conduced to this work.

REFERENCES

[1] C. F. García-Hernández, P. H. Ibargüengoytia-González, J. García-Hernández, and J. A. Pérez-Díaz, "Wireless sensor networks and applications: a survey," International Journal of Computer Science and Network Security, vol. 17, no. 3, pp. 264 –273, 2007.

[2] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn, "The rise of people-centric sensing," IEEE Internet Computing, vol. 12, pp. 12–21, July 2008.

[3] M. E. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC Sensor Web Enablement: Overview and high level architecture," in GeoSensor Networks, Second International Conference, GSN 2006, Revised Selected Papers, Lecture Notes in Computer Science, S. Nittel, A. Labrinidis, and A. Stefanidis, Eds, vol. 4540, Springer, 2008, pp. 175–190.

[4] H. Paulino and J. R. Santos, "A middleware framework for the Web integration of sensor networks," in Sensor Systems and Software - 2nd International ICST Conference, S-CUBE 2010, Revised Selected Papers, Lecture Notes of the ICST, Springer-Verlag, 2011, pp. 75–90.

[5] T. Kindberg, J. J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic, "People, places, things: Web presence for the real world," MONET, vol. 7, no. 5, pp. 365–376, 2002.

[6] ITU, "ITU Internet report 2005: The Internet of Things," International Telecommunication Union, Tech. Rep., 2005 [Online]. Available: http://www.itu.int/osg/spu/publications/internetofthings/

[7] L. Badger, T. Grance, R. Patt-Corner, and J. Voas, "Cloud computing synopsis and recommendations (draft), NIST special publication 800-146," Recommendations of the National Institute of Standards and Technology, Tech. Rep., 2011 [Online]. Available: http://www.nist.gov/itl/csd/20110512_cloud_guide.cfm

[8] A. Baptista, M. C. Gomes, and H. Paulino, "Session-based dynamic interaction models for stateful Web services," in Exploring Services Science - Third International Conference, IESS 2012, Lecture Notes in Business Information Processing, Springer-Verlag, 2012.

[9] M. Creeger, "Cloud computing: An overview," ACM Queue, vol. 7, no. 5, p. 2, 2009.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," Commun. ACM, vol. 53, pp. 50–58, April 2010.

[11] T. Kobialka, R. Buyya, C. Leckie, and R. Kotagiri, "A Sensor Web middleware with stateful services for heterogeneous sensor networks," in Intelligent Sensors, Sensor Networks and Information, 2007, ISSNIP 2007, 3rd International Conference, 2007, pp. 491–496.

[12] C. Stasch, A. C. Walkowski, and S. Jirka, "A geosensor network architecture for disaster management based on open standards." in Digital Earth Summit on Geoinformatics 2008: Tools for Climate Change Research, 2008, pp. 54–59.

[13] I. F. at al., "Modeling stateful resources with web services v. 1.1," Computer Associates International, Inc., Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation and The University of Chicago, Tech. Rep., 2004 [Online]. Available: http://www-106.ibm.com/developerworks/library/wsresource/ws-modelingresources.pdf

[14] OASIS, "Oasis web services resource framework (WSRF) TC," [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[15] F. Darema, "Dynamic data driven applications systems: New capabilities for application simulations and measurements," in International Conference on Computational Science (2), 2005, pp. 610–615.

[16] C. Gomes, O. F. Rana, and J. Cunha, "Extending grid-based workflow tools with patterns/operators," Int. J. High Perf. Comput. Appl., vol. 22, pp. 301–318, August 2008.

[17] V. Bhat, M. Parashar, M. Kh, N. K, and S. Klasky, "A self-managing wide-area data streaming service using model-based online control," in in Proc. 7th IEEE Int. Conf. on Grid Computing, 2006, pp. 176–183.

[18] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing degrees, models, and applications," ACM Comput. Surv., vol. 40, pp. 2–25, August 2008.

[19] M. Aldinucci, M. Danelutto, and P. Kilpatrick, "Towards hierarchical management of autonomic components: A case study," in Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP2009, IEEE Computer Society, 2009, pp. 3–10.

[20] K. Aberer, M. Hauswirth, and A. Salehi, "A middleware for fast and flexible sensor network deployment," in Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006, 2006, pp. 1199– 1202.

[21] F. Darema, "Dynamic data driven applications systems: A new paradigm for application simulations and measurements," In Int. Conf. on Computational Science, volume 3038, Springer LNCS, 2004.

Dr. Maria Cecília Gomes, PhD, is an Assistant Professor at the Departamento de Informática (Computer Science Department) of the Faculdade de Ciências e Tecnologia/Universidade Nova de Lisboa. She is a member of the Research Center for Informatics and Information Technologies (CITI), and member of the management committee and national PI of *ARTS: Towards Autonomic Road Transport Support Systems* (COST Action TU1102). Her current research interests include autonomic systems particularly applied to road transportation support systems, service-oriented computing, and models.

Dr. Hervé Paulino, PhD, is an Assistant Professor at the Departamento de Informática (Computer Science Department) of the Faculdade de Ciências e Tecnologia/Universidade Nova de Lisboa. He is a member of the Research Center for Informatics and Information Technologies (CITI), on which he is the principal investigator of the SABLE (Service Abstractions for Parallel Computing) research team, and also a collaborator member of the Research Center for Research in Advanced Computing Systems (CRACS). His current interests are on the fields of service-oriented computing and, concurrent and parallel programming.

Adérito Baptista, MsC, was a computer science master student at Faculdade de Ciências e Tecnologia/Universidade Nova de Lisboa. He is currently a software developer at Novabase, Portugal.

Filipe Araújo, MsC, was a computer science master student at Faculdade de Ciências e Tecnologia/Universidade Nova de Lisboa. He is a record management systems and R&D developer at Quidgest, Portugal.