Engineering Intelligent Sensor Networks with ASSL and DMF

Emil Vassev and Paddy Nixon
Lero–the Irish Software Engineering Research Center, University College Dublin, Ireland
emil.vassev@lero.ie, paddy.nixon@lero.ie

ABSTRACT

We describe the use of ASSL (Autonomic System Specification Language) and DMF (Demand Migration Framework) in the development of software systems for intelligent sensor networks. ASSL is used to formally specify and automatically generate autonomous intelligent sensor nodes. DMF is applied to connect those nodes in a sensor network. ASSL provides sensor networks with self-management behavior based on special policies allowing sensor nodes to reason and collaborate by exchanging information via a DMF instance.

KEYWORDS: sensor networks, autonomic systems, self-management, ASSL, DMF

1. INTRODUCTION

Nowadays, we witness how the latest in computing and communication technology emphasizes more and more low-cost sensor networks intended to help people in their daily lives. Such networks operate over sensors collecting and processing data in diverse domains such as air quality control, weather forecast, traffic control, security and surveillance applications etc. Although there have been great advances in the field of sensor networks [1, 9, 10, 11, 12, 13, 14, 15], the development of resource-efficient sensor networks able to adapt to situations in order to improve their efficiency is still a challenging task. Such a "smart" behavior requires "intelligent" sensor nodes able not only to sense the environment but also to reason and collaborate with other sensor nodes in the network. Such sensor networks (SNs) we term intelligent sensor networks (ISNs).

This research aims at building ISNs capable of self-management. We consider such systems to be *autonomic systems* (ASs) [2] employing self-management by virtue of special policies driving the network in question in critical situations. Conceptually, the AS paradigm draws inspiration from the human body's autonomic nervous system. The idea is that software systems can manage themselves and deal with dynamic requirements, as well as unanticipated changes, automatically, just as the human body does, through self-management based on high-level objectives [3]. Our approach to the development of ASs is ASSL (Autonomic System Specification Language), an initiative promoting formal specification, validation, and code generation of ASs within a framework [4, 5]. In order to build intelligent sensor nodes exhibiting AS features, we draw upon our experience[1] with the ASSL framework. Hence, with ASSL we specify and generate *intelligent sensor nodes*. To connect these sensor nodes in an ISN, we use a special networking mechanism called DMF (Demand Migration Framework) [8, 18]. Note that neither ASSL nor DMF were originally developed for the purpose of building ISNs, but the combination of both allows for this successful technological convergence applicable to heterogeneous sensor networks.

The rest of this paper is organized as follows. In Section 2, we review related work to intelligent networks such as 1) adaptable networks employing certain intelligent behavior; 2) energy-aware sensor networks employing energy-management algorithms; and 3) agent-based ISNs incorporating self-management features. In Section 3, we briefly

[1] With ASSL we successfully built prototypes of ASs such as the NASA ANTS [6] and NASA Voyager [7] missions.

978-1-4244-6622-1/10/$26.00 ©2010 IEEE

introduce the concept of sensor networks together with that of ASSL and DMF. In Section 4, we present our approach to the development of ISNs by using ASSL and DMF. In this section, we also present a case study demonstrating how our approach can be applied for developing an ISN for home-automation. Finally, Section 5 provides brief concluding remarks and a summary of future research and investigation trends.

2. RELATED WORK

One of the important aspects of any SN (sensor network) is the underlying network mechanism. By their nature, SNs are distributed networks with multiple nodes exchanging messages (cf. Section 3.1). Moreover, often network nodes can be used as re-transmitters and thus, there may be multiple routing paths used to deliver a message from a source to a destination. Here, as intelligent are considered special ad-hoc networks employing special adaptive routing protocols. Such networks decide on-the-fly the most appropriate route considering different factors such as: current network status, performance measures, cost of transmission over a given route, reliability of a path, time of transmission, etc.

Considerable work has been done on routing protocols in ad-hoc networks. For example, routing protocols for mobile wireless networks are discussed in [9, 10, 11, 12]. Another example is special routing algorithms based on game theory developed by Altman et al. [12].

Another aspect of SN intelligence is energy management. Practice has shown that energy efficiency appears to be of crucial importance for both performance and reliability of any energy-independent (battery-driven) SNs. Here, algorithms of energy awareness and management have been developed, where network intelligence is implemented at the level of single node or/and at the global level of the entire network [13].

An approach to ISNs providing autonomic behavior is described in [14]. Similar to our approach, there an ISN is achieved through the use of multi-agent architecture and self-management behavior.

In [15] an agent oriented programming paradigm for the development of intelligent sensor networks is presented. The proposed architecture for ISNs consists of autonomous intelligent agents that interact with other agents over special high-level communication protocol implementing a special declarative high-level agent communication language.

In our approach, we do not aim at efficient routing algorithms, although such can be implemented with ASSL as a global network-level behavior. Instead, by using ASSL we develop intelligent autonomous units embedding sensors and driven by self-management policies. Moreover, we may use ASSL to specify global self-management policies, those working at the network level and forming global network-level intelligence. Therefore, an ASSL-developed ISN usually employs an intelligent behavior at both levels – sensor node level and network level. Moreover, in our approach, the networking mechanism exposes a centralized topology and is independent of sensor nodes. This makes an ISN both reliable and efficient, because its network nodes are volunteers and any node can be easily replaced by new one, without interrupting the entire network.

3. PRELIMINARIES

3.1. Sensor Networks

In general, a sensor network is composed of sensor nodes connected to other sensor nodes. The network connection usually is wireless and sensor nodes often rely on routing protocols to communicate with other nodes not directly connected to the first. Usually, a sensor node has limited computational resources. This is due to the fact that in most cases, sensor networks rely on batteries, where high-performance hardware cannot be efficiently supplied with energy [1, 13].

In our approach, sensor nodes are considered to have enough computational power to run both a Java virtual machine and the Java-implemented self-management control software generated with ASSL. Note that ASSL generates Java code [4] and the employed DMS [8, 18] (a DMF instance) is JINI-based [16], which is a Java application as well.

3.2. ASSL

Although ASSL is dedicated to *autonomic computing* [2], with this work we demonstrate how it can be used for the development of sensor networks with self-management capabilities. In this subsection, we present the ASSL specification model by emphasizing special features that make the framework suitable for the development of ISNs.

3.2.1. ASSL Specification Model

ASSL is based on a specification model exposed over hierarchically organized formalization tiers. The ASSL specification model is intended to provide both infrastructure elements and mechanisms needed by an AS (autonomic system), or in this case by an ISN. Each tier of the ASSL specification model is intended to describe different aspects of the AS under consideration, such as special *service-level objectives, policies, interaction protocols, events, actions*, etc. This helps to specify an AS at different levels of abstraction imposed by the ASSL tiers (cf. Table 1).

Table 1. ASSL Multi-tier Specification Model

AS	AS Service-level Objectives	
	AS Self-management Policies	
	AS Architecture	
	AS Actions	
	AS Events	
	AS Metrics	
ASIP	AS Messages	
	AS Channels	
	AS Functions	
AE	AE Service-level Objectives	
	AE Self-management Policies	
	AE Friends	
	AEIP	AE Messages
		AE Channels
		AE Functions
		AE Managed Elements
	AE Recovery Protocols	
	AE Behavior Models	
	AE Outcomes	
	AE Actions	
	AE Events	
	AE Metrics	

The ASSL specification model considers the ASs as being composed of special *autonomic elements* (AEs) interacting over interaction protocols, whose specification is distributed among the ASSL tiers. Note that although ASSL allows for specification and code generation of interaction protocols, the latter cannot be used as an ISN networking mechanism, because ASSL currently does not generate distributed systems. Instead, it generates multithreaded systems with embedded messaging. Here, we rely 1) on ASSL to specify and generate sensor nodes in the form of AEs; and 2) on DMF to implement the needed networking mechanism, which connects the nodes together.

Table 1 presents the multi-tier specification model of ASSL. As shown, it decomposes an AS in two directions:

1) into levels of functional abstraction;
2) into functionally related tiers (sub-tiers).

With the first decomposition (cf. first column in Table 1), an AS is presented from three different perspectives depicted as three main tiers:

1) AS Tier forms a general and global AS perspective exposing the architecture topology, general system behavior rules, and global actions, events, and metrics applied to these rules.
2) ASIP Tier (AS interaction protocol) forms a communication perspective exposing a means of communication for the AS under consideration.
3) AE Tier forms a unit-level perspective, where an interacting sets of the AS's individual components is specified. These components are specified as AEs with their own behavior, which must be synchronized with the behavior rules from the global AS perspective.

It is important to mention that the ASSL tiers are intended to specify different aspects of the AS in question but it is not necessary to employ all of them in order to model an ISN. Thus, to specify a simple ISN, we need to specify a single AE per sensor node providing the self-management control software controlling the node's sensors and the communication with other AEs. Moreover, self-management policies must be specified to provide self-management behavior at the level of AS (the AS tier) and at the level of AE (AE tier) (cf. Table 1). Note that this rule is implied by the fact that any ASSL specification must be based on self-management [2].

With the following sub-subsection, we present some of the ASSL constructs needed to specify an ISN.

3.2.2. Self-management Policies

The self-management behavior of an AS (or ISN), is specified with ASSL self-management policies (cf. the appropriate tiers in Table 1). These policies are

specified with special ASSL constructs termed *fluents* and *mappings*:

- A fluent is a state where an AS enters with *fluent-activating events* and exits with *fluent-terminating events*.
- A mapping connects fluents with particular actions to be undertaken.

Self-management policies are driven by events and actions determined in a deterministic manner, similar to finite state machines. For the purpose of ISN development, self-management policies may be specified to control the network sensors and the process of sending and receiving messages. Moreover, both network-level (at the AS tier) and node-level (at the AE tier) self-optimizing policies can be specified. Self-management policies can be used to control the communication in real-time systems, which are bounded by deadlines. Deadlines may be a particular time, a time interval, or the occurrence of an event. Thus, we can use ASSL to specify real-time ASs, where different events can be used to trigger different policies intended to solve problems when the deadline cannot be met.

3.2.3. ASSL Events

ASSL aims at event-driven autonomic behavior. Hence, to specify self-management policies driving the sensor nodes of an ISN, we need to specify appropriate events. Here, we rely on the reach set of event types exposed by ASSL. To specify ASSL events, one may use logical expressions over service-level objectives, or may attach events to metrics (cf. Section 3.2.4), other events, actions, time, and messages. Moreover, ASSL allows for the specification of special conditions that must be stated before an event is prompted.

3.2.4. ASSL Metrics

For an ISN, one of the most important success factors is the ability to sense the environment and to react on sensed events. Here, together with the reachable set of events, ASSL imposes metrics as a means of determining dynamic information about external and internal points of interest. Although four different types of metrics are allowed [4], for the needs of ISN development, the most important are the so-called *resource metrics* intended to measure special managed resource quantities. Note that a *managed resource* (cf. Section 3.2.5) can be a controlled sensor and in such a case, an ASSL metric is linked with a network sensor.

3.2.5. Managed Resources

An AE (autonomic element) typically controls a managed resource specified with ASSL as *managed elements* [4]. Generally, a managed element (ME) is a functional unit (hardware or software). In an ASSL-developed INS, a managed element may present a controlled sensor (or a group of sensors). In order to understand how an ASSL-developed INS works, it is important to understand the AE-ME relationship. Note that an AE monitors and interacts with its MEs.

In ASSL, a ME is specified with a set of special interface functions intended to provide control functionality over the same. Thus, ASSL provides an abstraction of a ME through specified interface functions. Here, ASSL can specify and generate the interface controlling a ME, but not the implementation of this interface in that controlled ME. Here, when developing an ISN, the generated interfaces must be implemented by the appropriate sensor drivers.

3.3. DMF

DMF (Demand Migration Framework) [8, 18] is a generic scheme for migrating information in the form of messaging objects, in a heterogeneous and distributed environment determined by both information senders and information recipients (both termed as communication nodes). Hence, the framework establishes a context for performing migration activities, where the migrated messaging objects encapsulate behavior functions and data. In general, DMF may be applied as a concept underlying a generic architecture for the implementation of special family of Demand Migration Systems (DMSs) [8, 17, 18].

3.3.1. Rationale

Originally, DMF was developed in [8] to expose a generic framework for object-migration in a heterogeneous and distributed environment. By applying DMF, we can design a variety of DMSs conforming to a set of requirements described by the following elements [8]:

- *platform interoperability* – deals with process-machine boundaries and with the diversification of different hosting platforms (connects nodes running on Linux, Solaris, Windows, and Mac-OS platforms);

- *"at least once" delivery semantics* – ensures that no object could be delivered to a wrong recipient and is delivered at least once [3];
- *asynchronous communication* – communication nodes run independently and have no synchronized lifecycles;
- *no prioritization* – both objects and communication nodes are served equally by the system;
- *secure communication*;
- *fault-tolerant migration*;
- *hot-plugging* – communication nodes are "volunteers" in the communication process.

3.3.2. DMF Architectural Model for ISNs

DMF exposes a layer-structured architecture. Figure 1 depicts an ISN-elaborated variant of that architecture. As shown, the largest circle depicts an ISN as composed of AEs controlling sensor nodes. The double-lined inner circle depicts DMF. Here, AEs are communication nodes, and DMF is a communication intermediate between them.

DMF consists of two main functional layers called *demand dispatcher* (DD) and *migration layer* (ML) respectively. DMF relies on these two functional layers to form an asynchronous message-persistent communication protocol where messages are permanently stored and delivered upon request. DD (depicted by a bold-lined circle – cf. Figure 1) is an object-based storage mechanism able to dispatch messaging objects to their recipients. ML (depicted as a dark grayed layer on top of DD) is the layer performing object migration from an AE to another AE. ML makes the communication in a heterogeneous distributed environment possible.

In addition, ML (migration layer) emphasizes the use of special kind of agents called *transport agents* (TAs), which are based on distributed technologies exposed over a special TA interface. Therefore, in order to use the DMF communication protocol, the AEs (autonomic elements) of an ISN must adhere to the special TA interface exposed by the DMF transport agents. Moreover, the DD (demand dispatcher) layer establishes a context of demand propagator that consists of two layers - *demand space* (DS) and *presentation layer* (PL) (cf. Figure 1). The demand space layer defines a context of internal object-based storage mechanism. The presentation layer is an abstract layer on top of DS that makes all the DS functionality transparent and generic.

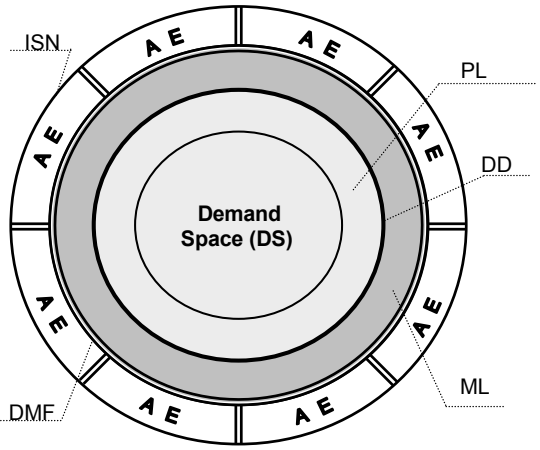

Figure 1. Demand Migration Framework (DMF)

3.3.3. DMS for ISNs

The communication protocol of an ISN is an instance of DMF termed DMS (Demand Migration System). In our approach, we rely on the DMS described in [17]. This DMS exposes transport agents based on JINI [16] and termed JINI TAs. Thus, the AEs controlling the sensors of an ISN adhere to the JINI TA interface.

4. BUILDING INTELLIGENT SENSOR NETWORKS

Our understanding of an ISN is that of a sensor network composed of sensors incorporating an event-driven behavioral mechanism to allow the network reacts to changes in the network structure (e.g., a node is down) or environment. As we have already explained in previous sections, we build an ISN by using ASSL to specify and generate intelligent sensor nodes as AEs. Next, we connect the generated AEs in a network by using a special DMS instantiated from DMF and exposing JINI TAs.

Figure 2 depicts a conceptual model of our network. As shown, the AEs controlling network sensors are connected in a network through a JINI-based DMS. Note that the network topology is centralized. The DMS stores the messages sent by AEs and delivers them to the recipient AEs when the latter are available. Moreover, in order to use the DMS, each AE connects to a JINI TA via a special interface.

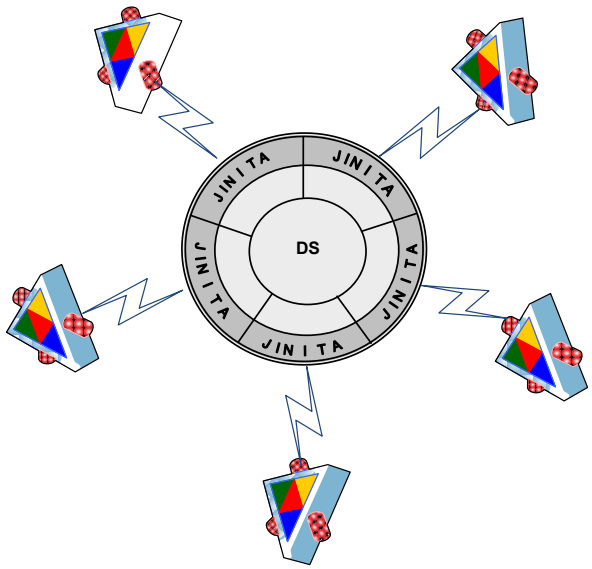

Figure 2. ISN's AEs Communicate via a DMS

Although this is not depicted in Figure 2, note that a single JINI TA can be shared by multiple AEs. However, to achieve better performance, it is recommended to use a distinct TA per AE.

4.1. Steps

The steps of building an ISN with ASSL and DMF are as follows:

1) Use ASSL to specify the AEs in terms of self-management policies (providing self-management behavior) and MEs (managed elements).
2) Generate the Java implementation of the AEs with ASSL.
3) Connect the generated AEs with the appropriate sensors through the generated ME interfaces.
4) Install the JINI DMS in place.
5) Connect the JINI TAs with the AEs through the generated ME interfaces.
6) Run the JINI DMS.
7) Run the ISN's AEs.

4.2. Case Study

In the course of this project we used ASSL and DMF to build an ISN for home automation. Our first step was to automate the living room of a house. We used ASSL to specify four different AEs composing the ISN for home automation:

- *Light AE* - controls the lights in the living room. This AE uses light sensors to determine the level of brightness in the living room and uses the light switcher to turn on/off the lights. Moreover, this AE communicates with the Motion AE to determine when and where motion is detected in the living room, which could prompt turning lights on.
- *Voice AE* - controls the microphones in the living room. This AE detects and recognizes speech. It communicates with the Light AE and with the Door AE to perform voice commands, such as "turn lights on/off" or "open/close door".
- *Motion AE* - controls motion detectors to sense the living room for motion. It zones the living room and detects where the motion is taking place and how many moving objects are there. The Motion AE communicates with the Door AE and with the Light AE.
- *Door AE* - controls (open or close) the door. It communicates with the Motion AE, e.g., when motion is detected towards door, the Door AE opens the door automatically.

4.2.1. ASSL Specification

For each one of these AEs we specified *self-management policies* and *managed elements*. The, ASSL specification is built around self-management policies providing intelligence. Recall that the self-management policies are driven by events and actions determined deterministically (cf. Section 3.2.2). The following ASSL fragments present a partial specification of the Voice AE.

```
AESELF_MANAGEMENT {
  OTHER_POLICIES {
    POLICY MANAGE_VOICE_COMMAND {
      FLUENT inSpeach {
        INITIATED_BY { EVENTS.speechDetected }
        TERMINATED_BY { EVENTS.commandRecognized,
                        EVENTS.commandNotRecognized }
      }
      FLUENT inCommandRecognized  {
        INITIATED_BY { EVENTS. commandRecognized }
        TERMINATED_BY { EVENTS.commandProcessed }
      }
....
      MAPPING { CONDITIONS { inSpeach }
                DO_ACTIONS { ACTIONS.recognizeCommand } }
      MAPPING { CONDITIONS { inCommandRecognized  }
                DO_ACTIONS { ACTIONS.processCommand } }
....  }}}
```

The first specification presents a self-management policy. This policy determines the behavior of the AE when a speech is detected. Here events are specified to initiate and terminate fluents within this policy. As shown, when speech is detected (via the controlled microphones) the AE starts this policy in an attempt to recognize a voice command. If such is recognized, it will be propagated to the Door AE (if the voice command is "open/close door") or to the Light AE (if it is "turn on/off lights") through the DMS installed in place by using a JINI TA (transport agent).

To control both the microphones and the JINI TA, the Voice AE specifies two managed elements determining the control interface. The following ASSL specification snippet presents the specified managed elements.

```
MANAGED_ELEMENTS {
  MANAGED_ELEMENT MICROPHONES {
    INTERFACE_FUNCTION speechDetected {
      RETURNS { Boolean } }
    INTERFACE_FUNCTION retrieveCommand {
      RETURNS { String } }
  }
  MANAGED_ELEMENT JINI_TA {
    INTERFACE_FUNCTION sentMessage {
      PARAMETERS { ISNMessage oMessage } }
    INTERFACE_FUNCTION receiveMessage {
      RETURNS { ISNMessage } }
  }
}
```

The specified interface functions help the Voice AE detects speech, retrieves a voice command from detected speech, and sends and receives messages. Note that the Light AE controls not only the sensors (microphones) but also the TA (JINI TA) allowing this AE to communicate through the DMS run in place. As shown, the MICROPHONES managed element is intended to detect and process speech. Here, for the implementation of the retrieveCommand interface function we rely on the Nuance Dragon NaturallySpeaking Solution, Version 9.0 [19]. Dragon NaturallySpeaking provides a speech to text engine with appropriate SDK for Java. As for the JINI_TA managed element, the Voice AE uses this to communicate with a JINI TA (transport agent) attached to this AE.

4.2.2. Implementation

We used the ASSL framework to validate the consistency of the ASSL-specified ISN and generate the implementation of the same. Here, the ISN system was generated as hierarchically organized Java packages nesting Java classes derived from the ASSL specification. The four AEs (*Voice*, *Light*, *Motion*, and *Door*) were generated with a special control loop that applies control rules specified and implemented as *self-management policies* (e.g., cf. MANAGE_VOICE_COMMAND in Section 4.2.1). The following Java code fragment presents an ASSL-generated control loop.

```
protected void controlLoop() {
  try {
//monitor-analyzer-simulator-executor
    oMonitor.perform();
    oAnalyzer.perform();
    oSimulator.perform();
    oExecutor.perform();
//applies self-management policies
    applyPolicies();
    Thread.sleep(tDelay);
  }
  catch ( InterruptedException ex )
  {....}
}
```

As shown, a special `controlLoop()` method is generated to handle special control loop calls (performed on a regular basis) for each AE. In its first part, the control loop uses four components (`oMonitor`, `oAnalyzer`, `oSimulator`, and `oExecutor`) to discover problems with ASSL-specified *metrics*. A detailed description of those four components refers to the autonomic computing nature of the generated code and is beyond the scope of this paper. The second part of this control loop is a function call of the `applyPolicies()` method that is intended to apply deterministically the self-management policies of an AE. The following code fragment presents a partial implementation of that method. This implementation is the same for all the four AEs generated for the ISN.

```
protected void applayPolicies() {
....
    //applies only "switched-on" policies
    if ( currPolicy.isSwitchedOn() ) {
      currPolicy.doAllMappings();
    }
  }
}
```

As shown, the policies of an AE (e.g., Voice AE) are applied by performing for each policy a `doAllMappings()` method. The latter maps actions to fluents and performs only those actions that are mapped to initiated fluents, e.g., as specified in the MANAGE_VOICE_COMMAND policy the inSpeach fluent is mapped to the recognizeCommand action (cf. Section 4.2.1). Note that a fluent is initiated if at least one of the fluent-initiating events has occurred, e.g., occurrence of the speechDetected event initiates the inSpeach fluent.

While performing, an action makes calls on the managed elements to do sensing or acting. For example, the ASSL-specified recognizeCommand action calls the sendMessage() method of the JINI_TA managed element to send messages (voice commands) to other AEs (Door AE or Light AE).

```
ACTION recognizeCommand { ...
  DOES {
    voiceCommand = CALL
AEIP.MANAGED_ELEMENTS.MICROPHONES.retrieveCommand;
    ....
    CALL AEIP.MANAGED_ELEMENTS. JINI_TA.sendMessage
(voiceCommand) ;
    ....
```

Here, managed elements such as MICROPHONES and JINI_TA (cf. Section 4.2.1) were generated as Java classes, where the interface functions are implemented as empty class methods. Thus, we had to complete those after the ISN code was generated. For example, we had to complete both sendMessage() and reciveMessage() methods of the JINI_TA class (generated for the identical managed element) by using the functionalities of the DMS. Thus, we programmed the JINI_TA class to connect to a JINI TA (transport agent) and use its functionality to send and receive messages.

4.2.3. Test Results

In this case study, we specified and generated the four AEs composing the ISN for home-automation. We also put together the generated AEs and the JINI DMS. However, we did not use real sensors. Instead, we simulated sensing home-automation environment with ASSL events specified to simulate sensor activity. Note that events trigger the specified policies by initiating appropriate fluents (cf. the AESELF_MANAGEMENT ASSL specification sample above). Thus, with such events we were able to simulate speech detection, voice command recognition and other sensor-related events. The test results demonstrated that, under simulated conditions, the run-time behavior of the ISN strictly followed the ASSL-specified self-management policies. Moreover, the ISN's AEs were able to exchange messages through the JINI DMS run in place.

It is important to mention, that we built an ISN for home-automation as a pure software solution, and thus we could not perform real sensor-based tests, but simulated ones. To perform real tests we need to implement both sensors and actuators in our ISN. The needed components such as motion detector (cf. Figure 3) or light control sensors can by bought online from ELV Elektronik Deutschland [20]. The use of real sensors will allow for further experimental evaluation. This will help us answer questions such as response time, efficiency of sensors and hardware control, etc. For example, a possible response time could be defined as the time it takes the system to open the door after a user has spoken the "open door" voice command.

Figure 3. PIR Motion Detector [20]

5. CONCLUSION AND FUTURE WORK

We have demonstrated how ASSL – a formal tool for development of autonomic systems, and DMF – a framework for message migration in distributed environments, can be used together to develop ISNs (intelligent sensor networks) with self-management capabilities. In our approach, we use ASSL to specify special behavioral policies provided by autonomic elements intended to control sensors via special managed elements. Here, we assume ISNs composed of autonomic elements specified with suitable ASSL specification constructs and whose Java implementation is automatically generated. Moreover, in our approach we use a JINI-based DMS (demand migration system) to connect the generated autonomic elements. This DMS provides a networking protocol needed by an ASSL-developed ISN, where the ISN's autonomic elements use special JINI TAs (transport agents) to communicate. As a proof of concept, we successfully built an ISN for home-automation, where sensors were simulated with special events.

Future work is concerned with further ISN experiments and development by including hardware attached to the control software generated by the ASSL framework. Moreover, we intend to build ISN prototypes incorporating self-management policies such as *self-healing*, *self-protecting*, and *self-adapting*. This will help us to investigate and develop

ISNs able to automatically detect and fix performance problems, e.g., by switching to alternative sensors.

ACKNOWLEDGEMENTS

This work was supported in part by an IRCSET postdoctoral fellowship grant (now termed EMPOWER) at University College Dublin, Ireland and by Lero – the Irish Software Engineering Research Centre.

REFERENCES

[1] D. J. Cook and S. K. Das (ed.), SMART ENVIRONNEMENTS: TECHNOLOGIES, PROTOCOLS, AND APPLICATIONS. John Wiley, New York, 2004.

[2] R. Murch, AUTONOMIC COMPUTING: ON DEMAND SERIES, IBM Press, Prentice Hall, 2004.

[3] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology", Technical Report, IBM T. J. Watson Laboratory, 2001.

[4] E. Vassev, TOWARDS A FRAMEWORK FOR SPECIFICATION AND CODE GENERATION OF AUTONOMIC SYSTEMS. DOCTORAL THESIS, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2008.

[5] E. Vassev and M. Hinchey, "ASSL: A Software Engineering Approach to Autonomic Computing", *IEEE Computer*, Vol. 42, No. 6, pp. 90–93, 2009.

[6] E. Vassev, M. Hinchey, and J. Paquet, "Towards an ASSL Specification Model for NASA Swarm-Based Exploration Missions", 23rd Annual ACM Symposium on Applied Computing (SAC 2008) – Autonomic Computing Track, ACM Press, pp. 1652–1657, 2008.

[7] E. Vassev and M. Hinchey, "Modeling the Image-Processing Behavior of the NASA Voyager Mission with ASSL", 3rd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'09), IEEE Computer Society, pp. 246–253, 2009.

[8] E. Vassev, GENERAL ARCHITECTURE FOR DEMAND MIGRATION IN THE GIPSY DEMAND-DRIVEN EXECUTION ENGINE, M. Sc. Thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2005.

[9] C. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers", *ACM SIGCOMM Computer Communication Review*, Vol. 24, No. 4, pp. 234–244, 1994.

[10] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad-hoc Wireless Networks", MOBILE COMPUTING, Vol. 353, Kluwer Academic Publishers, 1995.

[11] S. Das, C. Perkins, and E. Royer, "Performance Comparison of Two On-demand Routing Protocols for Ad-hoc Networks", Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000), IEEE Computer Society, pp. 3–12, 2000.

[12] E. Altman, T. Basar, T. Jimenez, and N. Shimkin, "Competitive Routing in Networks with Polynomial Costs", *IEEE Transactions on Automatic Control*, Vol. 47, No. 1, pp. 92–96, 2002.

[13] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy-aware Wireless Microsensor Networks", *IEEE Signal Processing Magazine*, Vol. 19, No. 2, pp. 40–50, 2002.

[14] D. Marsh, R. Tynan, D. O'Kane, and G. O'Hare, "Autonomic Wireless Sensor Networks", *Engineering Applications of Artificial Intelligence*, Vol. 17, No. 7, pp. 741-748, Elsevier, 2004.

[15] B. Karlsson, O. Bäckström, W. Kulesza, and L. Axelsson, "Intelligent Sensor Networks - an Agent-Oriented Approach", Workshop on Real-World Wireless Sensor Networks (REALWSN'05), Sweden, 2005.

[16] R. Flenner, JINI AND JAVASPACES APPLICATION DEVELOPMENT‖, Sams Publishing, Indianapolis, 2001.

[17] E. Vassev and J. Paquet, "A General Architecture for Demand Migration in a Demand-Driven Execution Engine in a Heterogeneous and Distributed Environment", 3rd Annual Communication Networks and Services Research Conference (CNSR'05). IEEE Computer Society, Halifax, Canada, pp. 176-182, 2005.

[18] E. Vassev, GENERAL ARCHITECTURE FOR DEMAND MIGRATION IN DISTRIBUTED SYSTEMS, LAP Lambert Academic Publishing, 2009.

[19] Nuance Dragon NaturallySpeaking Solution [website], 2009, http://www.nuance.com/naturallyspeaking/.

[20] ELV Elektronik Deutschland [website], 2009, http://www.elv.de/ output/controller.aspx.