

Integrating WSN Simulation into Workflow Testing and Execution

Duarte Vieira and Francisco Martins

Faculdade de Ciências da Universidade de Lisboa
LaSIGE & Departamento de Informática
Edifício C6 Piso 3, Campo Grande
1749 - 016 Lisboa, Portugal
dvieira@lasige.di.fc.ul.pt, fmartins@di.fc.ul.pt

Abstract. Sensor networks are gaining momentum in various fields, notably in industrial and environmental monitoring, and more recently in logistics. The information gathered from the environment (by sensor networks) may influence the execution of workflows, making it difficult to test these systems as a whole. Generally, the tests carried out on the aforementioned systems make use of recorded information in earlier workflow executions. Alternatively, we propose the testing of such workflows by incorporating results obtained from the simulation of sensor network applications, allowing the testing of new workflows, as well as of the changes made to a given workflow by events in the environment. This paper describes a means of integrating existing platforms with the aim of introducing the simulation of sensor networks in workflow testing and execution.

1 Introduction

A wireless sensor network (WSN) consists of a collection of tiny devices capable of measuring a given scalar or vector field and that can communicate wirelessly. In a WSN there can be nodes with additional capabilities (besides sensing the environment) such as act on the environment (actuators), collect and process data from sensors, or (re)configure the network behavior (base stations).

The topic of sensor networks has attracted the attention of both companies and research groups. The challenges raised in terms of hardware, such as device miniaturization, battery capacity improvement, and communication range increase, are as important as those raised at the software level, particularly in what concerns the operating systems and the programming languages for these devices. In both areas there have been important advances [2, 18, 27].

The applications of sensor networks are many and include, for example, reading our body's vital signs (body sensor networks) or monitoring environmental conditions (*e.g.*, management of air quality) [2]. Another application area is the *Internet of Things and Services* (ITS), which aims at integrating the *state of the world* seen from the *eyes* of sensors in high-level applications available on the *Internet*. An ITS application may benefit from environment observations and

adapt its behavior accordingly. For example, a home automation system may extend its features and, in addition to the traditional task scheduling (*e.g.*, turning on and off a given device according to a predetermined plan), react to environmental conditions or to behavioral rules of the inhabitants. Another application area is logistics, where, for example, we may want to tailor a delivery process according to the actual conditions of the goods being transported, and to the traffic information in the route to the destination. In this case, the information obtained from the sensors can lead to modifications in the delivery process, such as a change in the order by which the goods are delivered.

Applications that encode workflows are difficult to test because their behavior depends on external events (the world) that are, in the general case, nondeterministic. The most common approach to test these applications is to replay the execution of saved workflow traces. Although simple, this approach suffers from several problems, namely: (a) only allows the testing of workflows that retain the same execution trace as the ones saved, (b) disallows the testing of brand new workflows, and (c) prevents the testing of new variants of a given workflow. The approach we propose to test these applications is to obtain environmental information by simulating the sensor networks, enabling the creation of virtual environments where we can study the behavior of sensors *per se*, the sensor network as a whole, and the application. However, this approach comes with a price: (i) it requires the construction of a simulation model to simulate the WSN; (ii) as the scope of sensor networks widens it is necessary to use simulators, which constitutes by itself a challenge because simulating a sensor network with non-trivial behavior is far from being a simple, fast task; and (iii) the results of these simulations need to be integrated into the workflow management systems (WFMS).

In [26] we describe a process for automatizing the creation of simulation models for a WSN from high-level specifications that addresses (i) and (ii). In this paper, we focus on integrating these WSN simulations in workflow management systems, covering (iii).

We identify two main challenges in such integration: i) the communication between the WSN simulator and the WFMS, and ii) the changeover (in a transparent way) between using a WSN simulation and a real WSN. As for i) the communication needs to occur in both ways, *i.e.*, from the WSN simulator to the WFMS, when acquiring sensor readings, and *verso*, for example, for allowing the workflow to reprogram the network when necessary. In both directions we use techniques that push data between systems, instead of periodically querying the target systems. As for ii) we want to minimize the changes made in the workflow definition, and therefore reduce the testing impact of the changes caused by changeovers between real and simulated WSN.

In the remainder of this paper, Section 2 presents related work, specifically aimed at integrating real WSN data into workflow execution; Section 3 discusses the simulation of WSN, while presenting a programming language and a simulator; Section 4 presents some workflow management systems and elaborates on the integration of WSN simulations with the execution of workflows; Section 5

presents a scenario in logistics that illustrates the use of workflow management systems, integrating workflows with sensor networks to assess the conditions in which takes place a supply of materials sensitive to temperature; and, finally, Section 6 concludes the paper and presents some directions for future work.

2 Related Work

The integration between WSN and workflow execution present in the literature applies only to real sensor networks. Typically, the integration is performed when the data collected by the WSN is made available through web services. An example of such approach is the Graphical Workflow Execution Language for Sensor Networks [9] (GWELS), a language and toolkit for integrating WSN applications into workflow execution. The toolkit adapts WSN data coming from XML web services as data sources, and provides an environment where it is possible to define a system that tailors its behavior accordingly with the received data. Another example of this approach is presented in [10], where the authors propose to achieve integration in a fully standard-compliant way, using standard web service technologies for data acquisition and communication, and a Business Process Execution Language [15] (BPEL) engine for the workflow execution.

A similar approach in nature are the sensor grids [17] that aim at integrating WSN data (provided by web services) in grid computing environments, such as GridKit [11]. In [21], the authors use grid workflow management systems to integrate a WSN available on the grid in a workflow. The workflow code itself is generated from a graphical modeling environment.

In both previous approaches, the data is provided by *real* WSN. We argue that real WSN data is not suitable for testing workflows that encompass WSN applications, because it is generally not possible to control what the WSN senses. Data from WSN simulation, on the other hand, varies accordingly with the simulation itself and, therefore, gives us more control over the testing process. In addition, if the workflow requires the WSN to behave differently (to be reprogrammed), the previous approaches do not provide the means to test the reprogramming before deployment.

3 Simulation of Wireless Sensor Networks

WSN simulation provides a test bed that allows for quick deployments of WSN applications and gives the opportunity to test these applications in networks with a variable (potentially large) number of sensor nodes. When simulating WSN, we not only test low-level issues, such as signal transmission, or measurement of physical quantities, but also high-level aspects such as communication protocols and application execution. Simulation is therefore of great interest in any activity that involves sensor networks.

In [26] we propose a simulation model generator [26] for sensor network applications and present encouraging results, both in terms of simulation duration time

and in terms of memory usage when simulating applications with several hundred of sensors. In this section, we present Callas, a language for programming WSN, as well as the simulation model generator.

3.1 Callas

Callas [20] is a programming language aiming at establishing a formal basis for the development of languages and run-time systems for WSN. The language can be used directly as a programming language for sensor networks or as an intermediate language to which higher level WSN languages can be compiled.

The Callas programming language is type-safe. This property ensures that well-typed programs do not produce errors at run-time, a property of extreme importance in the context of sensor networks, in which testing and debugging are difficult or even impossible to perform after sensors are deployed in a real environment. Another feature of the language permits the reprogramming of sensors remotely. This allows for bug corrections or application upgrades without having to physically redeploy the sensors. In a Callas network, all network nodes implement the same interface (enforced at compile time), but they may behave differently depending on the actual implementation of the interface.

A Callas application consists of several files: an interface file defining the functions available to the sensor network (all nodes in the network share the same type); a program file by sensor type describing its behavior. and, finally, a network description file that details the network configuration in terms of, *e.g.*, number, type, position, behavior, of each sensor in the network. We omit Callas code in this paper, but the interested reader may refer to [19, 20] for several examples of programming with Callas.

The execution of Callas programs is performed using a virtual machine. This allows for abstracting sensors hardware (which is extremely heterogeneous) and for supporting features of the language, such as sensor reprogramming, across multiple platforms. Currently, we have an implementation of the Callas virtual machine (CVM) for the SunSpots, for sensors running TinyOS [12], and for the VisualSense [4] WSN simulator.

The architecture of the CVM is depicted in Figure 1. It includes three threads: one that runs the interpreter, one that receives messages from the network, and one that sends messages to the network. The communication model of the virtual machine is very akin to middleware systems except that calls are obviously asynchronous. The main thread executes the interpreter, that evaluates the Callas programs and reads and places messages from the input and on the output queues. There are two additional threads for interfacing between CVM and the sensor low-level communication devices, adapting messages (byte-code) accordingly.

The Callas language allows for low-level function calls to the sensor operating system through the use of a special language construct (**extern**). The CVM must implement the interface with the corresponding operations in the operating system. In fact, the CVM is parametric on an object, identified in the figure as *Ext Op*, that must be instantiated for the particular operating system where it

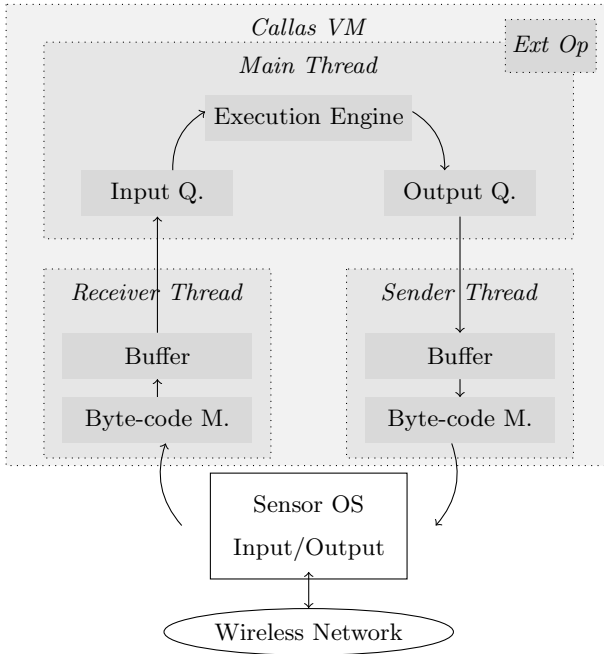


Fig. 1. The Callas virtual machine architecture

is running. For executing the CVM embedded as a VisualSense actor, we must provide this *Ext Op* object adapted to interact with VisualSense, in particular, with network-related code and with the external operations of the run-time system.

3.2 Simulating Callas Applications

Sensor network simulators can be categorized as *architecture-specific*, when only one node/architecture is supported, or as *generic*, when the simulator provides the means to model the sensor nodes. VisualSense [4] is a generic, Open Source WSN simulator based on the modeling framework Ptolemy II [8], developed at UC Berkeley. It allows (a) to simulate all the WSN aspects mentioned in the beginning of Section 3, (b) to simulate networks with nodes running different code among them, a rare feature [7], and (c) to model and simulate using a GUI. In Ptolemy II, modeling is accomplished using components called *actors* that interact solely by message passing, following the Actor Model [1].

The simulation model generator presented in [26] allows the end user to parameterize the number and distribution of the nodes, the program that runs on each sensor (or set of sensors), and the network and node models to be used. A simulation model for the application is generated from a Callas network description file. It contains information needed to compile the application (interface and program code for each sensor type), and information specific to construct

the simulation, given in the form of `key = value` pairs. Section 5 provides a concrete example of network description file used to create a simulation model.

The performance and scalability evaluation of the resulting WSN models execution, depicted in Figure 2 were obtained with VisualSense 7.01 on a Linux based PC with an Intel QuadCore 2.66GHz CPU and 3.4GB of RAM. Our experiments show that the simulation duration grows polynomially while the memory footprint grows linearly. We believe that simulation duration is not a critical factor, as one would expect to wait for a few hours before having results for a 5000 node network.

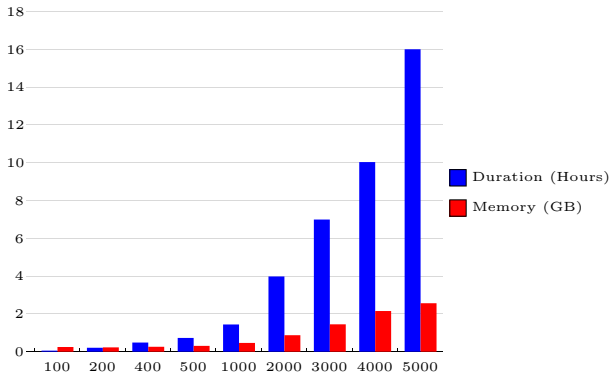


Fig. 2. Duration of simulation and memory usage (horizontal axis) given the number of sensors in the network (vertical axis)

4 Workflow Execution with WSN Simulation Integration

Workflow management systems are specialized computer systems where it is possible to model/program, test, and execute workflows. These systems are mainly used to analyze business processes, but they can also be used for design-time business process validation [22].

Most workflow management systems are bound to a particular language, for instance, the YAWL System [25] executes workflows written in Yet Another Workflow Language [24] (YAWL). There are also a number of workflow management systems that execute Business Process Execution Language [15] (BPEL), such as jBPM [14].

Typically, the integration of WSN into workflow execution is made with real WSN exposed as web services, or by integrating sensor networks in grid computing environments that can execute workflows. We claim that if real WSN data is to be used for testing workflows that integrate with WSN applications, then there is the need for an additional WSN test bed, since it is neither possible to control what the WSN senses, nor can we use the real WSN as a testing and as a production environment.

In this section, we devise a means of integrating WSN simulation in a workflow management system as a proof of concept of the integration of WSN simulation in workflow management systems with both-way communication. Section 4.1 explains how a WSN simulation can be integrated in a specific workflow management system, and Section 4.2 elaborates on the integration difficulties that stem from the lack of interoperability among workflow languages and execution engines.

4.1 Integrating WSN Simulation in the Kepler Workflow Management System

Kepler [3] is a scientific workflow management system, a system equipped with a structured set of operations over data sets. Other scientific workflow management systems include Taverna [13] and Triana [23].

Kepler supports GUI modeling and execution, workflow composition, distributed computation, and access to data repositories and web services. Like VisualSense, Kepler is a Ptolemy II specialization. In Kepler, the workflow execution is determined by a *computation domain*. For example, in the Synchronous Dataflow domain the execution is synchronous and occurs in a pre-calculated sequence; in the Process Networks domain the computation is parallel, meaning that one or more components may run simultaneously; and, in the Discrete Event domain, workflow execution is triggered by events and takes time into account.

One may use the same components interchangeably in Kepler and VisualSense. Therefore, it should be possible to include simulation models from VisualSense in Kepler workflows, obtaining a means of acquiring from the sensor network simulation into the workflow execution. Furthermore, since we are able to generate VisualSense simulation models from Callas applications, we can ease the simulation-related work for the user. We use this configuration to prove the feasibility of the integration of WSN simulation in workflow management systems.

In practical terms, the integration is as follows: the VisualSense WSN model is wrapped in a workflow component. The wrapper component provides an interface that allows both to acquire data from the WSN and to parameterize it. More complex WSN to workflow interfaces may be defined using the GUI.

The communication between the workflow and the WSN simulation is mediated by the model wrapper. It converts the messages from the workflow to the network, and vice-versa, allowing for both way communication. The wrapper may be reconfigured, even at execution time, in order to allow a different interaction. The WSN integrated in the workflow (whether simulated or real) may be switched to another one, even at execution-time. This allows for testing WSN application changes made by workflows in testing environments (WSN simulation) before deployment (real WSN).

Other difficulties of integrating VisualSense's sensor network models in Kepler may lie in the (possible) computation domain heterogeneity. WSN simulation is performed in the Wireless domain, an extension of the Discrete Event domain that is not suited for all types of workflows. However, there should be no difficulties in the type of business processes that we are interested in simulating, because they are usually event oriented.

4.2 Workflow Interoperability

A way to mitigate integration difficulties, and to use WSN simulation models in more workflow management systems is to create robust mechanisms of workflow interoperability, *i.e.*, the ability to execute workflows in a distributed way, using two or more distinct workflow management systems. The available variety of workflow management system engines and description languages hinders workflow interoperability. For example, Taverna [13] interprets the Simple Conceptual Unified Flow Language [13] (SCUFL), Kepler uses the Modeling Markup Language [5] (MoML), Yawl system [25] uses Yet Another Workflow Language [24] (YAWL), and Triana [23] interprets, in addition to its proprietary format, the Business Process Execution Language [15] (BPEL). Summing to the aforementioned difficulties (the variety of description languages and of execution platforms), workflow specification languages usually have different degrees of expressiveness, making the translation among them not always possible, which compromises interoperability by language translation.

Workflow interoperability could be achieved by standardizing the specification/execution language. There has been such attempts, for example, the Workflow Management Coalition created XPDL [15], and Microsoft and IBM have created BPEL, both aiming to become *the standard*. Another way to achieve interoperability of workflows is by integrating workflow engines so that it is possible to run each workflow in its execution environment, but being able to interact with other workflows, running in other environments. Such an approach is proposed by Kukla et al. [16]. The authors see workflow management systems as (legacy) applications embedded in a Grid Computing environment, in the case, GEMLCA [6] (Grid Execution Management for Legacy Code Applications).

5 Use Case: The Vaccines Delivery Process

The following scenario describes a workflow process integrated with environmental readings obtained from a WSN. A company distributes vaccines by several of its customers (*e.g.*, pharmacies, hospitals). The vaccines are very sensible to the environment temperature and are compromised if exposed to a temperature above its accepted parameters. To cater for this specificity, vaccines are transported in small containers equipped with its own cooling and monitoring systems. The vehicle is itself supplied with a general refreshing system that maintains the overall temperature dependent on the total cargo.

The control vehicle system consists of a sensor network including the temperature sensors of each container and a base station connected to the GSM system of the vehicle that is used for communications with the head office. Each sensor is programmed to fire an alert message should the temperature of a container reaches a given threshold (that might be different for each container). The base station is responsible for communicating the alert messages to the head office, for defining the GPS delivery routing, and for managing the overall cooling system.

The delivery workflow process is controlled centrally at the company's head office and is in contact with the vehicle's control system. In case the vehicle

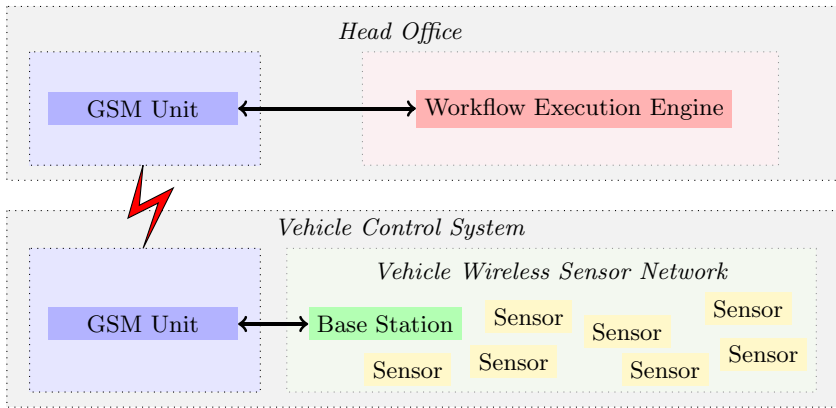


Fig. 3. The vaccine's delivery configuration

reports an alert indicating that some vaccine container is in danger of deteriorate, it triggers alternative workflows, like changing the delivery order of the goods, that, in turn, may originate changes in the vehicle's control system. The changes could be, for instance, the communication of a new destination coordinates for the GPS system, or the transmission of a new algorithm for adjusting the vehicles overall temperature, caused by the download of cargo from the vehicle.

Figure 3 depicts the scenario configuration, with the head office workflow execution system and the vehicle's control system communicating via GSM. Also it illustrates the local communications among the GSM units and the workflow execution engine and the vehicle sensor network, which is organized as a base station and several sensors.

Testing this workflow using the traditional approach requires a method to replay the communications with the truck. However, this is not required if the simulation of sensor networks is integrated with the workflow execution. Furthermore, the integration may allow the use of simulation data from sensor networks in richer scenarios than the one just described.

From the WSN application (defined in the network file, `network.calnet`, depicted in Figure 4), a simulation model, depicted in Figure 5 is automatically generated. The model can be further edited in VisualSense.

The network description file allows to specify the number of nodes (`size`) and their positions (`position`), the network and node models (`template`) (the persistent format of Ptolemy II has the `.moml` extension) and the communication range (`range`).

The two temperature sensors, `Node1` and `Node2`, in blue, execute the code in `node.callas`. Periodically, they send the temperature in each container to the base station, represented in green. The base station executes the code in `sink.callas`. It is possible to identify the node communication ranges, as well as their relative positions.

```

# file: network.calnet
interface = iface.caltype

sensor :
  code = node.callas                                # code for sensor in container
  size = 2                                          # two sensors of this type
  range = 50                                       # sensor range
  position = random 0, 0 to 10, 10                # distributed randomly
  template = containerSensor.moml                 # sensor simulation model

sensor :
  code = sink.callas                                # code for the base station
  size = 1                                          # one base station
  range = 50                                       # base station range
  position = explicit 0, 0                          # positioned at a particular place
  template = gsmSink.moml                          # base station simulation model

template = containerNetwork.model                 # network model
    
```

Fig. 4. A Callas network description file with simulation parameters

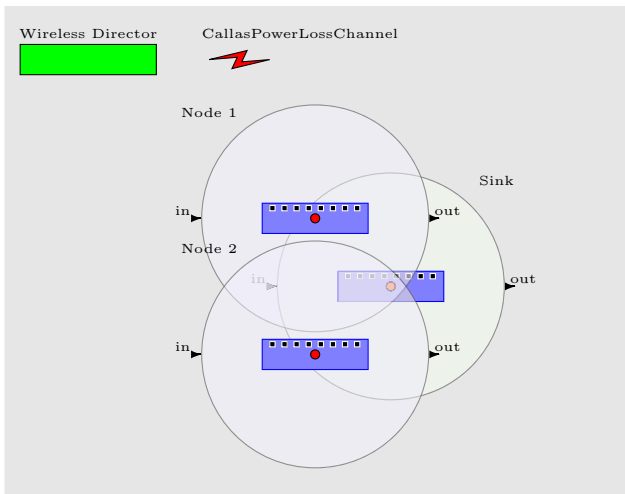


Fig. 5. Sensor network as defined in VisualSense

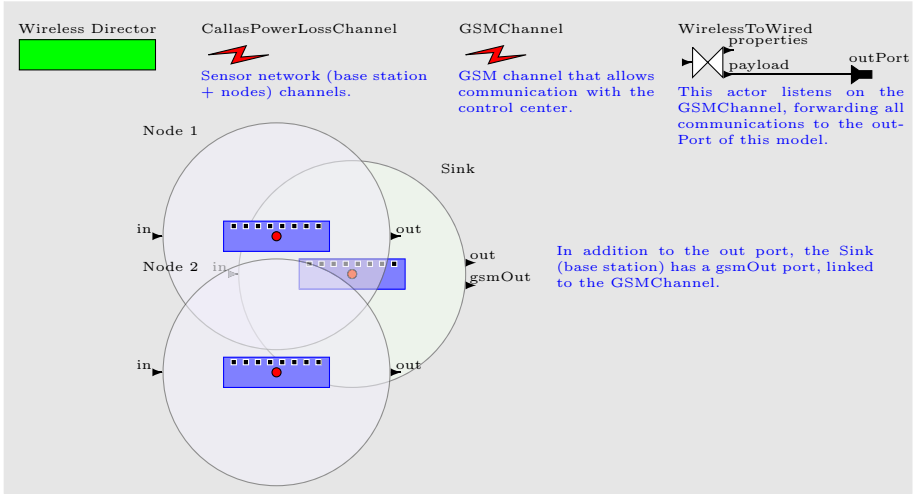


Fig. 6. VisualSense WSN simulation model to be integrated in the Kepler workflow

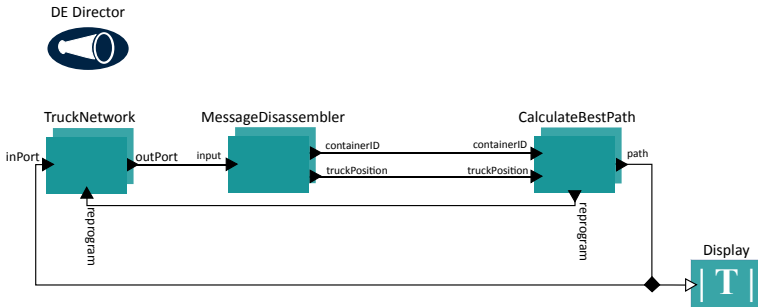


Fig. 7. Workflow in Kepler. Actor TruckNetwork encapsulates the network model in Figure 6. The CalculateBestPath actor encapsulates the workflow of path recalculated.

The communication between the sensor nodes and the base station is made through a channel (CalasPowerLossChannel). This simulates the wireless communication between the sensors in the container and the base station, taking into account signal properties such as power loss, and avoids the transmission of repeated messages. Ports in/out of all the represented nodes receive/send messages through CalasPowerLossChannel. The communication of the base station with the control center is simulated by the GSMChannel. Notice that the base station is the only node that can send messages on GSMChannel, namely output messages, making use its gsmOut port.

The workflow described in Figure 7 calculates the best path for the deliveries, taking into account the container temperatures, and the truck location. The

sensor network integration is made by encapsulating the network model in a Kepler actor (`TruckNetwork`), that acts as a data source. The workflow execution is, in this case, triggered by a message received from `TruckNetwork`. In a different workflow, the execution could be started by another event, possibly being altered by an incoming message from the sensor network. In the top right corner of Figure 6, there is the `WirelessToWired` actor, that receives the messages sent through `GSMChannel` and dispatches them through the `outPort` of `TruckNetwork`, depicted in Figure 7. In the workflow integration, this message is routed to a `MessageDisassembler` actor that extracts the `containerID` and `truckPosition` values, needed for the `CalculateBestPath` workflow. It should be mentioned that the `WirelessToWired` actor (that connects the sensor network to the workflow) is independent of the network and of the workflow; it is only a message broker.

6 Conclusions and Future Work

This paper presents an approach to integrate sensor network simulations into the execution of workflows, which allows the testing of higher level applications based on information from the *things* in the world. Our proposal aims to integrate the simulation of sensor networks of VisualSense [4] in the Kepler workflow management system [3], exploiting the interoperability of components (actors) between the two systems. This approach is feasible because both systems are extensions of the Ptolemy II modeling and simulation platform [8]. For the creation of WSN simulation models we make use of a generator (of simulation models) that we presented previously [26]. We believe that our approach is a valuable contribution for testing applications (not only those based on workflows) that depend on environmental values. It enables the execution of new workflows using simulated data, and eases experimentations with new scenarios. In contrast, testing workflows based on information from previous executions does not seem as flexible and complete.

As for future work, our initial focus will be on the validation of the proposed integration model, as well as obtaining results that allow us to evaluate the proposed solution. An interesting point that deserves further attention is workflow interoperability, *i.e.*, the ability to execute workflows in a distributed way, using two or more distinct workflow management systems.

Although we have not addressed the availability of sensor information via web, this does seem viable and we envision what it could be achieved using web services: the base stations would have to be identified (and named) in the network description file, so that they could be presented as webservice, and so that the clients can register themselves in order to be notified by the base stations.

Acknowledgments. The authors are partially supported by project CALLAS from Fundação para a Ciência e Tecnologia (PTDC/EIA/71462/2006).

References

1. Agha, G.: *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge (1986)
2. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. *IEEE Communications Magazine* 40(8), 102–114 (2002)
3. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S.: Kepler: An extensible system for design and execution of scientific workflows. In: *SSDBM 2004: Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, pp. 423–424. IEEE Computer Society, Los Alamitos (2004)
4. Baldwin, P., Kohli, S., Lee, E.A., Liu, X., Zhao, Y.: Modelling of sensor nets in Ptolemy II. In: *Proceedings of IPSN 2004*, pp. 359–368. ACM Press, New York (2004)
5. Brooks, C., Lee, E.A., Liu, X., Neuendorffer, S., Zhao, Y., Zheng, H.: Heterogeneous concurrent modeling and design in java (volume I: Introduction to ptolemy ii). Technical Report UCB/EECS-2008-28, Electrical Engineering and Computer Sciences University of California at Berkeley (2008)
6. Delaitre, T., Goyeneche, A., Kacsuk, P., Kiss, T., Terstysanszky, G., Winter, S.: Gemlca: Grid execution management for legacy code architecture design. In: *EUROMICRO 2004: Proceedings of the 30th EUROMICRO Conference*, pp. 477–483. IEEE Computer Society, Los Alamitos (2004)
7. Egea-Lopez, E., Vales-Alonso, J., Martinez-Sala, A., Pavon-Mariño, P., Garcia-Haro, J.: Simulation Scalability Issues in Wireless Sensor Networks. *IEEE Communications Magazine* 44(7), 64–73 (2006)
8. Eker, J., Janneck, J.W., Lee, E.A., Liu, X., Liu, J., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity - the ptolemy approach. *Proceedings of IEEE* 91(2), 127–144 (2003)
9. Glombitza, N., Lipphardt, M., Werner, C., Fischer, S.: Using graphical process modeling for realizing soa programming paradigms in sensor networks. In: *Proceedings of the 6th Annual IEEE/IFIP Conference on Wireless On demand Network Systems and Services*, pp. 61–68 (2009)
10. Glombitza, N., Pfisterer, D., Fischer, S.: Integrating wireless sensor networks into web service-based business processes. In: *MidSens 2009: Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*, pp. 25–30. ACM, New York (2009)
11. Grace, P., Coulson, G., Blair, G., Mathy, L., Yeung, W., Cai, W., Duce, D., Cooper, C.: GRIDKIT: Pluggable overlay networks for grid computing. In: Chung, S. (ed.) *OTM 2004*. LNCS, vol. 3291, pp. 1463–1481. Springer, Heidelberg (2004), doi:10.1007/978-3-540-30469-2_40
12. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. In: *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 93–104. ACM Press, New York (2000)
13. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. *Nucleic Acids Research* 34(Web Server issue), 729–732 (2006)
14. JBoss: jbpmp website, <http://www.jboss.org/jbpmp/> last accessed in October 20, 2010)
15. Ko, R., Lee, S., Lee, E.: Business process management (bpm) standards: A survey. *Business Process Management Journal* 15(5) (2009)

16. Kukla, T., Kiss, T., Terstyanszky, G., Kacsuk, P.: A general and scalable solution for heterogeneous workflow invocation and nesting. In: *WORKS 2008: Proceedings of the Workflows in Support of Large-Scale Science, Third Workshop*, pp. 1–8. Springer, Heidelberg (2008)
17. Lim, H., Teo, Y., Mukherjee, P., Lam, T., Wong, W., See, S.: Sensor grid: integration of wireless sensor networks and the grid. In: *The IEEE Conference on Local Computer Networks, 30th Anniversary*, pp. 91–99 (2005)
18. Lopes, L., Martins, F., Barros, J.: *Middleware for Network Eccentric and Mobile Applications*, ch. 2, pp. 25–41. Springer, Heidelberg (2009)
19. Lopes, L., Martins, F.: A semantically robust framework for programming wireless sensor networks. TR 2010–01, DCC/FCUP (March 2010), <http://www.dcc.fc.up.pt/dcc/Pubs/TReports/TR10/dcc-2010-01.pdf>
20. Martins, F., Lopes, L., Barros, J.: Towards the safe programming of wireless sensor networks. In: *Proceedings of ETAPS 2009* (2009)
21. Reddy, A., Kumar, A., Janakiram, D.: Workflow process model integrating wireless sensor networks with grids. Technical report, Distributed and Object Systems Lab, Dept. Of CSE (2007)
22. Rozinat, A., Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., Fidge, C.J.: Workflow simulation for operational decision support. *Data Knowl. Eng.* 68(9), 834–850 (2009)
23. Taylor, I.J., Schutz, B.F.: Triana - A Quicklook Data Analysis System for Gravitational Wave Detectors. In: *Second Workshop on Gravitational Wave Data Analysis*, pp. 229–237 (1998)
24. van der Aalst, W.: Yawl: yet another workflow language. *Information Systems* 30(4), 245–275 (2005)
25. van der Aalst, W.M.P., Aldred, L., Dumas, M., ter Hofstede, A.H.M.: Design and implementation of the YAWL system. In: Persson, A., Stirna, J. (eds.) *CAiSE 2004*. LNCS, vol. 3084, pp. 142–159. Springer, Heidelberg (2004)
26. Vieira, D., Martins, F.: Automatic generation of WSN simulations: From Callas applications to VisualSense models. In: *Proceedings of SENSORCOMM (2010)* (to appear)
27. Yoneki, E., Bacon, J.: A survey of wireless sensor network technologies: Research trends and middleware’s role. Technical Report UCAM-CL-TR646, University of Cambridge (2005)