Demo Abstract: From Business Process Specifications to Sensor Network Deployments

F. Casati[‡], F. Daniel[‡], G. Dantchev[†], J. Eriksson[*], N. Finne[*], S. Karnouskos[†], P. Moreno Montero[**], L. Mottola[*], F.J. Oppermann[+], G.P. Picco[‡], A. Quartulli[‡], K. Römer[+], P. Spiess[†], S. Tranquillini[‡], T. Voigt[*]
[**]Acciona Infraestructuras S.A. (Spain), [†]SAP AG (Germany), [*]Swedish Institute of Computer Science,
[+]University of Lübeck (Germany), [‡]University of Trento (Italy),

***Abstract*—The industrial adoption of wireless sensor networks (WSNs) is hampered by two main factors. First, there is a lack of integration of WSNs with business process back-ends. Second, programming WSNs is still challenging as it is mainly performed at the operating system level. To this end, we provide `make`*Sense* – a unified programming framework and a compilation chain that, from high-level business process specifications, generates code ready for deployment on WSN nodes.**

I. Introduction and Application Scenarios

Wireless Sensor Networks (WSNs) are small, untethered computing devices equipped with sensors and actuators. WSNs can be easily deployed and are able to self-organize to achieve application goals. Research has made significant progress in solving WSN-specific challenges such as energy-efficient communication. Industry, however, is reluctant to adopt WSNs. We believe this is due to two unsolved issues, integration and unification.

Integration refers to the need for strong cooperation of business back-ends with WSNs. Current approaches typically consider the WSN as a stand-alone system. As such, the integration between the WSN and the back-end infrastructure of business processes is left to application developers. Unfortunately, such an integration requires considerable effort and significant expertise spanning from traditional information systems down to low-level system details of WSN devices. Moreover, these two sets of technologies satisfy very different goals, making the integration even harder. This paper presents a holistic approach where application developers "think" at the high abstraction level of business processes, but the constructs they use are effectively implemented in the challenging reality of WSNs.

Unification refers to the need for a single, comprehensive programming framework. It is notoriously difficult to realize WSN applications. They are often developed atop the operating system, forcing the programmer away from the application logic and into low-level details. The many programming abstractions existing [1] are hard to use since they typically focus on one specific problem. To drastically simplify WSN programming, particularly for business scenarios, we need a broader approach enabling developers to use several abstractions at once. In this demo, we showcase a unified comprehensive programming framework where existing WSN programming abstractions can blend smoothly.

A paradigmatic example of our target scenarios is ventilation in buildings. Fans are commonly operated at a fixed rate, independent of room occupation, resulting in unnecessary ventilation of unoccupied rooms and over-ventilation of sparsely occupied ones, ultimately wasting energy. A smarter strategy may consider room occupation, resulting in sustainable building management. Consider an office environment, where employees book meeting rooms on the Web through a back-end process notifying the expected participants. Room ventilation is minimal when no meeting is scheduled. Sensors and actuators driven by the business process increase ventilation before the meeting and until human presence is detected or CO_2 levels are above threshold.

Realizing this system requires a tight integration between the business process and the network of sensors and actuators dispersed in the environment, as the application logic needs to extend to the latter. Moreover, implementing the processing for adaptive ventilation complicates application development, as it departs from traditional data collection most common in WSN to encompass possibly distributed control loops.

II. Approach

Our design revolves around three fundamental goals:

- `make`*Sense* must *seamlessly integrate* with existing business process technology, providing an adoption path that complements, instead of disrupting, existing methodologies and technologies with WSN ones.
- `make`*Sense* must be *modular* and *extensible*. As we aim for our system to be useful across several real-world applications, extensibility is key to ensure that the programming abstractions and their implementation can be easily adapted to the specificity of the target domain as well as to unforeseen needs.
- `make`*Sense* must *self-optimize* w.r.t. high-level performance goals. This ability to self-adapt is necessary to support long-lasting, operational business processes immersed in the physical environment and subject to the vagaries of wireless communication.

These goals are directly reflected in the `make`*Sense* architecture which is based on the separation of concerns provided by a distinction in layers of functionality: *i)* an *application* layer concerned with business processes and their modeling; *ii)* a *macroprogramming* layer concerned with the distributed execution of activities within the WSN; *iii)* a *run-time* layer concerned with the low-level aspects supporting the above and enabling self-optimization.

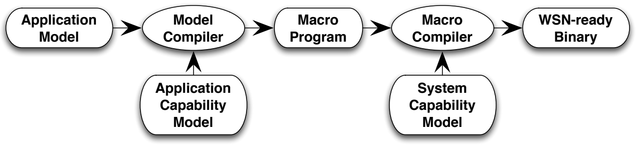

Fig. 1. Compiling business process models into WSN-executable code.

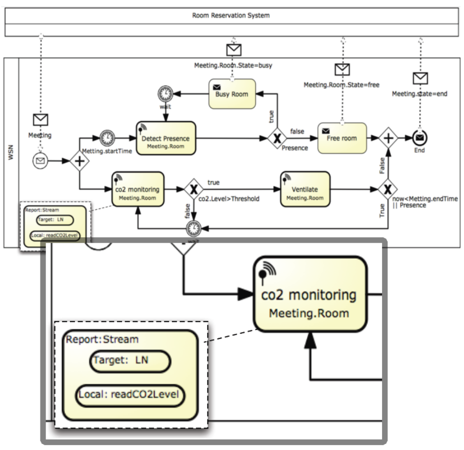

Fig. 2. BPMN diagram for a fragment of the ventilation scenario.

A model-driven approach connects the three layers (Figure 1). Using an extended verion of the Business Process Model and Notation (BPMN), the application model represents a holistic, network-agnostic view of the entire business process, i.e., including the WSN and the process back-end. It includes performance requirements (e.g., a certain level of reliability, or a minimum lifetime).

The semantic link among layers is achieved by two compilation steps. The model compiler takes as input the application model and an application capability model. The latter is a coarse-grained description of the WSN, providing information such as the type of sensors/actuators available and their operations. The model compiler translates these descriptions into a program written in a macro-programming language, serving as an intermediate language closer to the reality of WSN systems, yet high-level enough to be potentially used directly by a developer. The macro language is similar to Java, but offers static memory management and threading optimized for motes. It integrates several existing programming and networking abstractions such as Logical Neighborhoods [2] to specify a set of nodes or collection tree protocols into a common object-oriented framework that is based on the notion of Actions whose execution can be configured by embedded declarative languages as illustrated in the next section. The macro compiler takes as input the macro-program generated by the model compiler and a system capability model. The latter provides finer-grained information on the deployment environment (e.g., how many sensors of a given type are deployed at a location). The macro-compiler generates executable code that relies only on the basic functionality provided by the run-time support available on the target nodes. By leveraging the system capability model, the macro compiler can generate different code for different nodes, based on their application role.

```
...
code nhoodTemplateS = {:
  neighborhood template CO2Sensors()
    f.getFunction() = "sensor" and t.getType() = "co2" :};
code sensorNeighborhoodDef = nhoodTemplateS + {:
  create neighborhood co2Sensors from CO2Sensors () :};

Target co2Sensors = lnew LN();
co2Sensors.instantiate(sensorNeighborhoodDef);

Report co2Stream = lnew Stream();
co2stream.setTarget(co2Sensors);
co2Stream.setParameter("period", 5 * 60);
co2Stream.execute();
...
```

Fig. 3. Macro-programming language fragment for Figure 2.

III. Case Study

Figure 2 depicts a fragment of the business process model for the ventilation scenario discussed above. The whole process is modeled with two participants, the WSN-aware participant on top and the intra-WSN participant (modeled in more detail) that is converted into an application by generating macrocode. The zoomed part of the process shows a WSN activity that sets up and executes a periodic reading of CO_2 sensors in a certain room. A *target* identifies a set of nodes satisfying application constraints, and gives the ability to apply a distributed action to the nodes in this set.

By graphically combining abstractions (here, a *target* to specify the room and a *local action* to read the sensor are placed inside a *report* action to collect the sensor readings), setting all necessary parameters, and using meta-information of the current WSN setup, the model becomes rich enough to be transformed into the macrocode in Figure 3.

This code describes the instructions to define a *target* including all CO_2 sensors and to collect periodic data from them using an instance of *report* action implementing a `Stream` concrete abstraction. Specifically, the abstraction-specific code inside the `code` variable is the Logical Neighborhood [2] custom language. This is used to create an instance of *target*, referring to local actions to retrieve the function and type of node to possibly include in the target. The *target* is given as parameter to a `setTarget` method invoked on an instance of *report*. The remaining method invocations are used to set parameters for the functioning of the `Stream` instance, e.g., its reporting period.

The BPMN model may also contain application performance objectives. Based on this and monitoring data, the self-optimization functionality tunes the protocols' parameters, e.g., by going into a very low power mode when no meeting is scheduled and no presence of people has been detected.

Acknowledgments. This work is supported by the European Commission through the projects make*Sense* (www.project-makesense.eu) and CONET (www.cooperating-objects.eu).

References

[1] L. Mottola and G. Picco, "Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art," *ACM Computing Surveys*, vol. 43, no. 3, 2011.

[2] ——, "Logical Neighborhoods: A Programming Abstraction for Wireless Sensor Networks," in *Proc. of the Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, 2006.