Easy Programming of Integrated Wireless Sensor Networks

D7.4 – Publishable Summary Report

Grant Agreement no: 258351
www.project-makesense.eu

Date: July 11, 2014

Author(s) and affiliation: Thiemo Voigt, Luca Mottola, Anneli Avatare Nöu (SICS), Kay Römer (UZL), Gian Pietro Picco (UNITN), Patrik Spiess (SAP), Patricio Moreno Montero (ACCIONA)

Work package/task: WP7 – Project Management
Document status: Final
Dissemination level: Public
Keywords: Wireless sensor networks, business processes, integration, dissemination, exploitation

Abstract This document is the Publishable Summery Report within the Project Final Report for the FP7 `make`*Sense* project, representing formally the deliverable 7.4 as defined in Annex 1 (Description of Work).

Disclaimer

Contents

List of Figures

List of Tables

1 Executive summary

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 for two main reasons. First, there is a lack of integration of WSNs with business process modeling languages and 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.

To achieve these goals, the project has devised a three-layer architecture based on the separation of concerns. The application layer is concerned with business processes and their modeling. At this layer we extended a BPMN editor with WSN specific constructs and provided a model compiler that compiles the extended BPMN model to the intermediate make*Sense* macroprogramming language. The macro-programming layer constitutes an extendable framework that in contrast to existing solutions can encompass several WSN programming abstractions can also be used stand-alone. The macro-compiler compiles the macro-programming code to binary code that is executed by a sensor network. The sensor network integrates with business process execution engines via messages. The make*Sense* modelling layer allows application developers to specify performance objectives that can change during the run-time of the system. This is necessary to support long-lasting business processes subject to changing requirements. Therefore, the make*Sense* run-time is designed to self-optimize towards the performance objectives.

The project has deployed a pilot application in a student residence in Cadiz, Spain. In the students' rooms, a CO_2 sensor measures the CO_2 level and if the measured is below a pre-defined threshold an actuator placed in the bathroom's ceiling opens a flap to ventilate the air. The installed system worked correctly keeping the CO_2 threshold below the pre-defined value.

We have evaluated the make*Sense* system from many different points of view. The end-user perspective has shown that the make*Sense* system is cheap, easy to deploy and flexible in that its functionality can easily be extended. Our life cycle cost analysis shows significant savings compared to conventional wireless (ca. 40%) and wired systems (ca. 66%). Implementing the same applications in the make*Sense* framework and in plain Contiki demonstrated that make*Sense* applications require less code, the code is more flexible and easier to adapt but requires, as expected, more memory than native Contiki applications. User studies with several groups have shown that the make*Sense* framework successfully delivers what it has been designed for. This makes make*Sense* an attractive tool that enables domain experts to develop integrated sensor network applications for industrial and societal benefit. To our knowledge, there is no competitive offering, that combines all the aspects of make*Sense*. Concepts from make*Sense* have already been incorporated into the SAP HANA Platform, IoT Edition.

2 Project context and objectives

2.1 Motivation

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, schematically shown in Figure 2.1. Theses issues also lead to higher total cost of ownership than necessary.

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. We design 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. Many programming abstractions exist [1], but 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. We design a unified comprehensive programming framework into which existing WSN programming abstractions can blend smoothly.

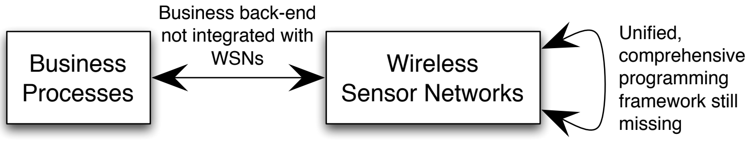
Business back-end not integrated with WSNs
Business Processes
Wireless Sensor Networks
Unified, comprehensive programming framework still missing

Figure 2.1: Open problems for using WSNs in business processes.

Solving the problem of integration and unification would allow also domain experts which are not low level programmers to develop integrated sensor network applications that span from business processes to sensor networks. Example applications in that area are numerous. The business scenarios that have driven our work are *(i)* ventilation on demand, *(ii)* a solar energy scenario where make*Sense* technology could help to verify that a solar panel has not lost its efficiency as well as simplify monitoring and maintenance tasks and *(iii)* condition-based maintenance for vessels. During the project, we performed a deployment in a student residence in Cadiz, Spain, that implement a ventilation-on-demand scenarios where CO_2 sensors readings were used to trigger ventilation when the CO_2 value in a room was above a pre-defined threshold.

2.2 Contributions

Towards this end, we have made the following main contributions:

1. Towards the problem of integration, we have extended a popular business process specification notation (BPMN) with WSN-specific constructs. The editor we developed makes WSN application development accessible also to domain experts.

2. Towards the problem of unification, the `make`*Sense* macroprogramming framework allows programmers to use several programming abstractions within the same program. These abstractions provide the key concepts that enable interaction with the WSN. Their composition can be achieved by using common control flow statements, provided by a core language that serves as the "glue" among macroprogramming abstractions.

3. Further, we have performed a life cycle cost analysis showing that the `make`*Sense* approach significantly reduces the total cost of ownership of a building automation system compared to a conventional WSN installation and a conventional wired system. The total cost of ownership would be about 180.000 Euro for a wired system. A conventional wireless sensor network system would lower the costs already to 105.000 Euro while a system developed with the `make`*Sense* approach would be disposable for a mere 60.000 Euro.

We verified the achievements of the major objective of the `make`*Sense* project to simplify the development of sensor network applications using dedicated user studies. The estimation of the total cost of ownership has been based on ACCIONA's experience from previous projects.

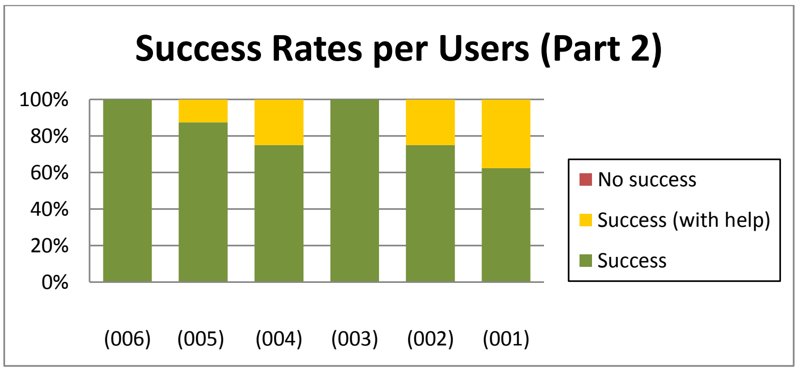

Figure 2.2: Success Rates in Study

At the business process modeling level, our extended BPMN editor enabled an exploratory study with a semi-realistic setting consisting of a teaching and a testing phase. After some

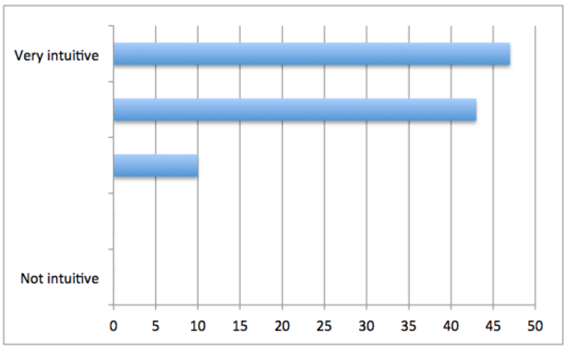

Figure 2.3: Use: do you find the concept of macroprogramming in makeSense intuitive?

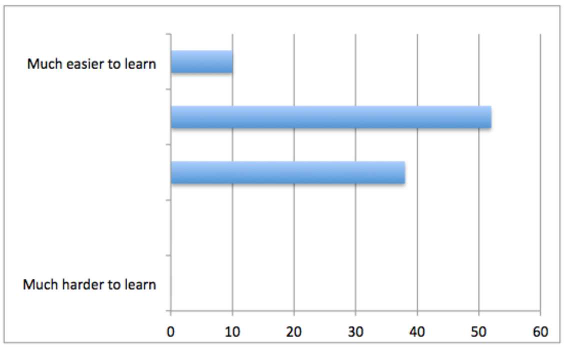

Figure 2.4: Feedback: compared to programming with Contiki/C, is makeSense easier to understand and use?

introduction and one learning exercise task, the success rate of the participants in the second part was generally very high as Figure shows 2.2.

In a second user study, we gauged the effectiveness of the makeSense macroprogramming framework when used by novice sensor network programmers. The students participating in this study were first given a 1.5 hour general introduction to sensor networks, covering challenges, applications, programming, networking, and hardware platforms at a fairly high level. Next, we taught the students programming with the bare-bone Contiki operating system and the C language. The following day, the students were taught the makeSense macroprogramming framework.

This study showed that the concept of macroprogramming, which is known to cause confusion for novice sensor network programmers because of the many flavors available in the literature [1], appears nonetheless fairly intuitive in the case of makeSense (Figure 2.3). Compared to programming in plain Contiki/C, makeSense is perceived as greatly simplifying the implementation of sensor network functionality (Figure 2.4). This is the key comparison we are interested in, and the results confirm that we have achieved the goal of unification through an extensible framework that indeed simplifies sensor network application development.

3 A description of the main S&T results/foregrounds

The makeSense project provides a number of technical contributions across different aspects, from the high-level application modeling using extended business process notations, down to the run-time mechanisms needed for efficient execution on resource-constrained sensor nodes, through dedicated sensor network macroprogramming abstractions. Together, all these technical contributions lead to a system that is cheaper to design, develop, deploy and maintain than traditional wireless sensor networks and a wired system. After presenting an overview of makeSense from an end-to-end perspective, we briefly describe next the major results of the project across all levels.

3.1 makeSense End-to-end Vision and Run-through

3.1.1 End-to-end Vision

Wireless sensor networks enable many interesting applications such as environmental and process monitoring, asset tracking and building automation. Many of these applications can benefit from an integration with business processes. Examples include ventilation of rooms and predictive maintenance.

Developing wireless sensor network applications is currently performed using low level C programming, which is known to be a difficult and time-consuming task only mastered by experts. Further, the integration with the business back-end is programmed manually. The makeSense project takes a new, different approach. It provides a graphical editor where domain experts can develop integrated sensor networking applications in BPMN (Business Process Model and Notation). This approach enables the design and development of such applications in one tool, allowing domain experts to design integrated applications and generate the code both for the business process back-end and the sensor network in the same tool. Since the final executable code is auto-generated, the need for testing decreases as auto-generated code should contain less errors compared to code written by an embedded software developer [2]. Furthermore, when changes in the application are required, the changes can be made directly in the application model within the graphical editor, the code can generated again and finally installed on the sensor nodes using the makeSense code dissemination tools, all without the need of expensive external services.

By avoiding wires, wireless sensor networks inherently simplify the actual deployment and disposal of monitoring and control applications compared to wired systems. In addition to the benefits of contemporary wireless sensor networks, the makeSense approach eases the final testing and quality control, which becomes less expensive than for conventional wireless sensor network approaches since in makeSense the code is auto-generated. We have estimated that make*Sense* drastically decreases the total cost of ownership of integrated sensor networks compared to conventional sensor networks (with a cost reduction of more than 40 %) and wired systems (where the cost reduction if roughly 66 %).

3.1.2 Example Run-Through

Let us take a look at a specific applications scenario, the automatic ventilation of rooms, for example, student dormitories or meeting rooms. Reducing the time the ventilation is on saves energy and cost. An external business process in this case is the room reservation system that would record the periods of time when rooms are not reserved and avoid ventilation during that time. In the future, an external business process could help to avoid ventilation during times when energy prices are high and preferably schedule ventilation when energy prices are low which would further reduce the ventilation cost. The sensor network would monitor occupancy of the room, measure CO_2 and schedule ventilation when the CO_2 readings are below a threshold. In fact, the make*Sense* final deployment has consisted of such a system that we deployed in a student dormitory in Cadiz, Spain [3] showing the benefits of make*Sense*.

Using make*Sense*, a domain expert can design the business process in BPMN using the make*Sense* graphical editor. A BPMN model for our deployed ventilation on demand application is shown in Figure 3.3. As discussed above this design includes both the business process and the wireless sensor networking application and we have shown that with little training domain experts are actually able to perform this task. Once the design is finished, the executable WSN code is auto-generated and can be installed on the sensor nodes, while the rest of the process can be interpreted by a regular BPMN execution engine, that can interface further business back-ends. Our life cycle cost analysis has shown that compared to a wired application or a convectional wireless sensor network approach, the make*Sense* approach is substantially cheaper, reducing the cost for design as well as development and testing by roughly 30-40% as discussed in more detail in Section 3.7. This cost saving is significant since development and testing constitute the most expensive phase for wireless sensor systems (see Figure 3.13). The reduced cost for testing stems from the auto-generation of code.

Conventional wired approaches are very expensive when it comes to deployment and disposal, as the wires and the infrastructure for them need to be set up. Hence, wireless approaches are drastically cheaper than wired ones. Once a system is running, it is often recognized that functionality needs to be added or slightly changed. Using the make*Sense* approach, the domain expert can go back to the graphical editor, modify the BPMN diagrams, re-generate the WSN code, and re-install the updated code on the sensor nodes. The make*Sense* approach reduces the maintenance cost by around 20% compared to conventional wireless sensor networks.

In summary, by enabling domain experts to graphically design and develop wireless sensor networks applications, by avoiding the need for wires, by reducing testing, and by making it simple to change the applications, make*Sense* drastically reduces the cost of ownership of integrated sensor networks applications. To give an impression on the cost reduction, we have calculated the total cost of ownership of a ventilation on demand system as described above using today's market prices. While it would be about 180.000 Euro for a wired system, a conventional wireless sensor network system would lower the costs already to 105.000 Euro while a system developed with the make*Sense* approach would be disposable for a mere 60.000 Euro.

3.2 Architecture

In this section we first outline the application domain supported by make*Sense*, followed by a high-level description of the make*Sense* architecture shown in Figure 3.1. Here, we split the description in three parts, representing the phases characterizing the development of a make*Sense* application: its design based on business process modeling constructs, the generation of the application code optimized for the scenario at hand, and its deployment in the final execution environment.

3.2.1 Applications

The make*Sense* system has been specifically designed to support and simplify the development of applications that involve business processes involving i) traditional computer systems, ii) wireless sensor networks, and iii) human actors. We give two example applications to outline the application domain supported by make*Sense*.

The first application example is *ventilation on demand*, which is about ventilation systems for buildings. Today, it is common practice to run those ventilation systems at a fixed rate. From an energy management perspective, however, it would be desirable to minimize ventilation to the amount that is necessary, because this would minimize the energy consumption of the ventilation system itself. Additionally, at times of high or low outside temperature, reduced ventilation leads to reduced need for cooling or heating the buildings.

As a specific instance of this general design space, we consider *meeting room ventilation*. The general idea is to ventilate meeting rooms only prior and during scheduled meetings and have them run at minimal or no ventilation for the rest of the time. In order to support more flexible meeting schedules the system also senses the presence of persons in the room and continues to ventilate if a meeting runs longer, or shuts down ventilation if a meeting has ended prematurely.

There are several human and technical actors participating in the use case. A meeting organizer schedules the meeting in the meeting room by entering the meeting information into a reservation system. Meeting participants receive the invitation to the meeting and arrive at the start time of the meeting. Wireless sensor nodes detect both CO_2 level and the presence of persons in the room with sensors, like passive infrared sensors. All information is used to pre-ventilate the meeting room before the meeting takes place. This is useful to remove bioeffluents that might have become concentrated while the room was in standby ventilation mode with reduced or no ventilation. The ventilation rate is adjusted by air flaps in the ventilation systems that are controlled by wireless actuator nodes. If the CO_2 level is below a given threshold (e.g., due to low occupancy), ventilation could be reduced even during meetings. When no presence of persons is detected and the CO_2 level is in the normal range, ventilation can go back to standby mode, even if the meeting was scheduled longer than it actually was taking place.

As a second example, we consider *predictive maintenance* which is a form of maintenance based on tracking the trends of several physical quantities that may provide insights in prospective failures of the machinery under control. By identifying the risk of potential breakages early and without stopping or dismantling the machine, one may follow the evolution of the defect until it really constitutes a danger, as well as schedule shutdowns, maintenance operations, and the supply of spare parts. This would ultimately reduce the repair time and be more economically effective.

One specific instance of this class of applications is monitoring the engine conditions aboard sea cargo vessels, using WSNs to monitor relevant physical quantities on the engines, such as vibrations, engine oil and ambient temperature, and rotation regimes. Sensed data is fed as input to predictive maintenance algorithms. Depending on the type, severity, and probability of the potential breakage detected, the maintenance algorithms may: *i*) automatically take some pre-established corrective actions, e.g., reduce the operating regime of the engines to avoid a fail-stop condition, and simply notify the chief engineer aboard the vessel; or *ii*) notify the chief engineer that a prospective breakage is detected but no immediate corrective is possible.

The chief engineer monitors the machine operation and all actions taken on it, being them automatic or manual. In case *ii* above, he may decide to reconfigure the WSN to gather more information on the prospective breakage. Such action may involve changing the operation of the WSN software, e.g., increasing the sensing rate to obtain finer-grained information. Should addressing the prospective breakage require changing a part of the engine, the chief engineer first checks if the spare part is aboard the vessel. If not, he notifies the area director on-shore of the necessary parts.

The area director checks if the necessary parts are found in the company warehouse. If so, he triggers the shipping of the part to a location decided to trade off the cost of shipping (e.g., by plane or by truck), against the risk of a fail-stop condition on the vessel should the breakage worsen, the labor cost at different maintenance ports, and the company losses involved in slowing the delivery of the goods aboard the vessel. The latter may be a concern especially in case the goods are perishable. In case the required parts are not found at the company's warehouse, the area director notifies the general director that an order needs to be placed and shipped to the vessel.

3.2.2 Design: Business Process Modeling Revisited

As illustrated in Figure 3.1, in make*Sense* the design of the business process is carried out by an *application modeler* by using a modeling language which is an extension of the standard Business Process Modeling Notation (BPMN). BPMN provides constructs for describing process activities, their decomposition into smaller activities, and the control flow governing their interaction. In make*Sense*, we extend BPMN with constructs expressly designed to model the salient characteristics of the WSN. As a result, the application modeler can focus, with the uniform mindset enabled by a single notation, on the "standard" business process activities hand-in-hand with those directly concerned with the WSN, by producing an *application model*. The latter is produced by the modeler by means of a dedicated tool which extends an existing open-source BPMN editor.

Nevertheless, because of the peculiarity of WSNs, the BPMN-based application model alone is not enough to describe the target environment. A conceptual view of the basic functionality made available by the WSN is necessary, as this constitutes the basic blocks WSN-specific business activities build upon. The *application capability model*, also specified by the application modeler possibly after consulting with domain experts, contains this information as a set of application-level attributes and operations made available by the WSN nodes. Example of attributes are the room where a node may be located, or the sensors available to nodes. Note that the application capability model does *not* specify the actual values of these attributes, but only their names and types. Moreover, the application capability model specifies the set

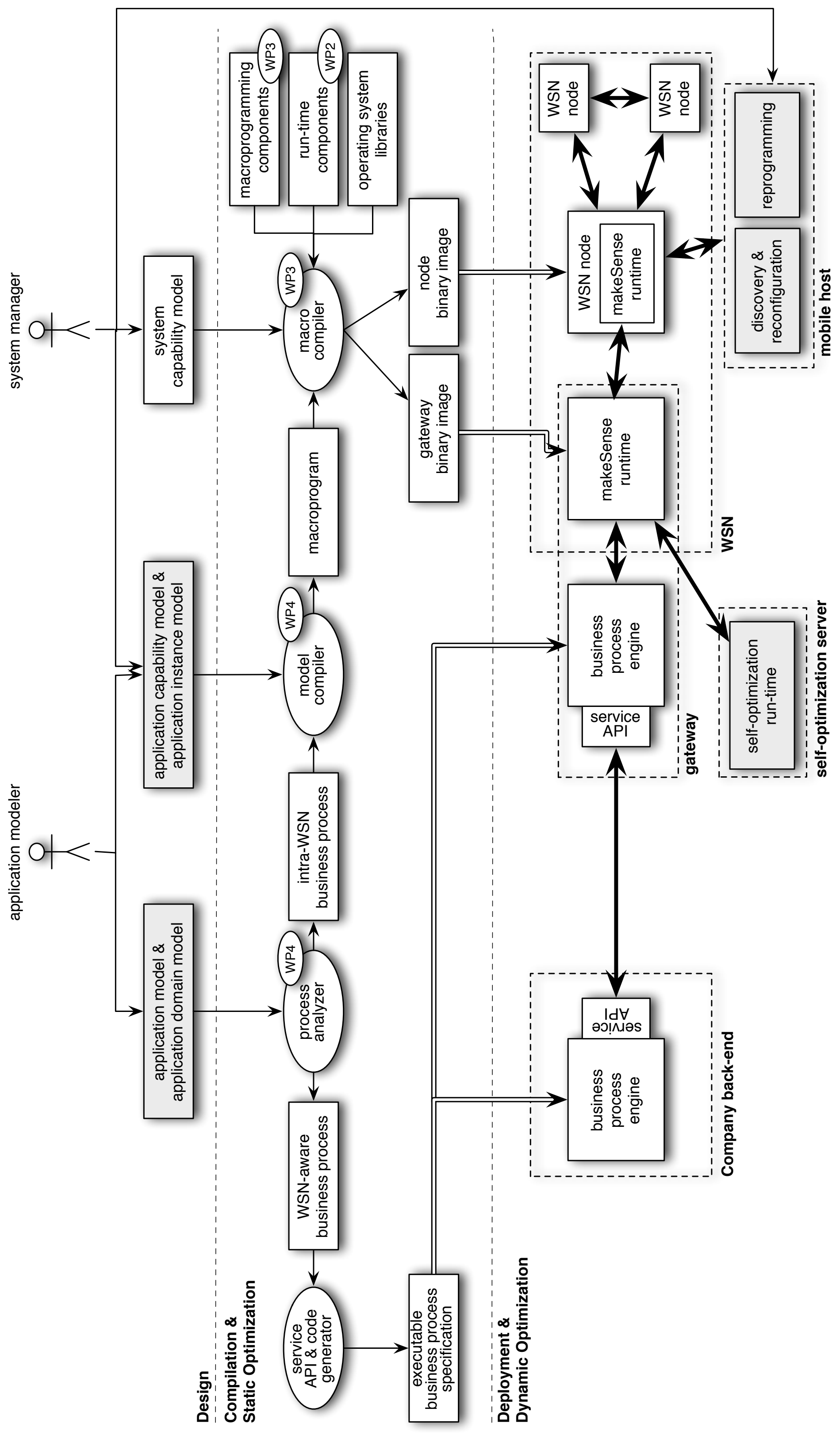
system manager
application modeler
Design
Compilation & Static Optimization
Deployment & Dynamic Optimization
system capability model
application capability model & application instance model
application model & application domain model
macroprogramming components
WP3
run-time components
WP2
operating system libraries
macro compiler
WP3
macroprogram
model compiler
WP4
intra-WSN business process
process analyzer
WP4
WSN-aware business process
service API & code generator
executable business process specification
node binary image
gateway binary image
WSN node
WSN node
WSN node
makeSense runtime
makeSense runtime
WSN
business process engine
service API
gateway
reprogramming
discovery & reconfiguration
mobile host
self-optimization run-time
self-optimization server
service API
business process engine
Company back-end

Figure 3.1: The make*Sense* architecture.

of operations available, e.g., as simple as reading the sensor value or as complex as providing on-board filtering and smoothing of the sampled data. The actual parameter values for a given deployment are contained in the *application instance model*.

The application capability model, along with the application model, provides a complete description of the high-level behavior of the application. However, it is not sufficient to provide the system-level parameters necessary for the generation of the corresponding code, its optimization, and finally its deployment. These include information about the physical properties of the nodes (e.g., amount of memory available) and of the network encompassing them (e.g., 1-hop vs. multi-hop). This information typically requires a second actor, the *system manager*, who has knowledge of the WSN deployment and characteristics, and is able to describe the appropriate information in a *system capability model*, which is used during the compilation process as described next.

3.2.3 Compilation: The `make`*Sense* Toolchain

The development toolchain transforms the *application model* representing the business process into an executable format. Its constituents are:

Process analyzer The business process can be regarded as composed by two main sub-processes: one concerned directly with behavior of the WSN (the *intra-WSN business process*, and one (the *WSN-aware business process*) mainly concerned with the pure business logic but nevertheless aware of, and able to interact with, the former. The task of the process analyzer is, given the application model, to draw the line between the two business processes, which follow different paths in the toolchain. The intra-WSN business process is fed into the WSN-specific tools developed in `make`*Sense*, while the WSN-aware business process is transformed in executable form using standard tools.

Service API & code generator Transforms the *WSN-aware business process* model into executable form, and generates the appropriate service interfaces enabling communication to and from the WSN, supporting both data and control flow. Execution of this process part takes place by using a business process execution engine capable of executing BPMN 2.0.

Model compiler Translates the *intra-WSN business process* model into a macroprogram, described by means of a macro-programming language. The latter essentially serves as a high-level intermediate language bridging business processes and WSNs.

Macro compiler Translates the macroprogram received as input into deployment-ready binary images, targeting both the WSN nodes and the gateway (if present). The macrocompiler weaves together the components directly supporting the macroprogramming constructs, which in turn rely on the lower-level components provided by the run-time. The translation is driven by the directives contained in the system capability model, which allows the compiler to optimize the code images for the specific hardware and network topology.

3.2.4 Deployment & Dynamic Optimization

Figure 3.1 shows also how the artifacts produced by the toolchain are deployed within the system. Indeed, the business process is executed in a distributed manner between the *company back-end*, where the bulk of the business process infrastructure resides, and the WSN.

Communication between the two is enabled by a dedicated machine, the *gateway*, which is conceptually "closer" to the WSN. The gateway machine, which can be thought of as a standard or embedded PC, hosts two software components. The first ("make*Sense* runtime") acts as the "access point" to the WSN and enables, for instance, injecting queries as well as actuation and reconfiguration commands. The second component ("business process engine"), instead, is still conceptually part of the business process. At a minimum, this second component acts as a sort of proxy, relaying requests from the company back-end towards the WSN, and vice versa. Requests and replies are communicated using standard business process technology (e.g., SOAP), and are in turn translated into lower-level commands to the make*Sense* WSN run-time on the same machine, through a dedicated protocol. However, in some scenarios the business component may be more complex than a proxy, and actually contain a portion of the business process executed in a distributed fashion close to the WSN.

The components generated by the macrocompiler are deployed directly on WSN nodes. Their performance is optimized based on the high-level directives (e.g., privilege data yield vs. lifetime) provided in the application model. These percolate through the entire toolchain to provide optimized binary code. In some cases, the performance directives can be expressly changed by the business process, for example because the process is engaged in a particular activity that demands a different performance goal. Moreover, WSNs are often characterized by dynamicity, for example, changes in the wireless connectivity and node failures. Therefore, the binary image deployed on nodes must be able to *self-optimize* to adapt to changing environmental or process conditions. A *discovery and reconfiguration* tool enables localized interaction with WSN nodes for dynamic discovery and setting of their attributes, operated from an in-field mobile host. Finally, a *selective reprogramming* tool allows the operator to modify at run-time the program deployed on some or all nodes of the WSN, either from an in-field mobile host (as shown in Figure 3.1) or from the gateway.

3.3 Business Process Modeling

Based on the meta abstractions and the macroprogramming language described in the previous chapter, we further wanted to simplify the programming of WSNs by offering graphical modeling tools. Domain experts with no programming skills but basic knowledge of the modeling standard *Business Process Modeling Notation (BPMN)*, a de facto standard for process modeling, should be enabled to sketch processes that are later refined by a more technical role. The resulting process model is then compiled through the chain of model compiler and macro compiler and deployed to run on a given WSN.

As Figure 3.2 depicts, the modeling is supposed to be carried out by two roles: 1. the BPMN domain expert, a person who knows both basic BPMN and the business or technical scenario the WSN is supposed to support and 2. the BPMN technical expert who knows BPMN and the make*Sense* abstractions in detail. Ideally, when a domain expert has modeled several processes, s/he could become more familiar with BPMN details and the make*Sense* abstractions so that with time the two roles can be enacted by one person.

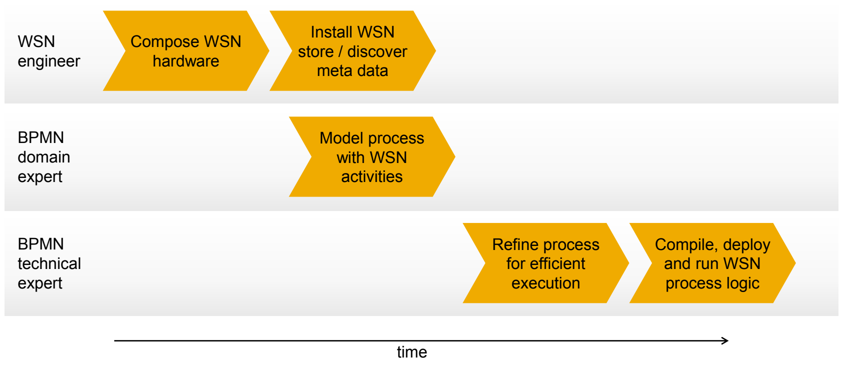
WSN engineer
Compose WSN hardware
Install WSN store / discover meta data
BPMN domain expert
Model process with WSN activities
BPMN technical expert
Refine process for efficient execution
Compile, deploy and run WSN process logic
time

Figure 3.2: `make`*Sense* modeling process

When integrating WSNs with business processes, most research projects and productive set-ups merely add a service facade to the WSN and orchestrate its services centrally. If middleware is deployed, that is done either purely in a central system [4] or with additional local components close to the WSN or on its gateway [5]. In makeSense, we used a more radical approach. As our goal was to further decrease the effort of programming WSN applications, tools for process modeling are used to create the application top-most level. A process modeler models hybrid processes, of which one part is executed conventionally in a central execution engine, while another part is executed directly by the WSN.

The chosen process modeling standard BPMN comes with a diagram model, where the semantics of each artifact is defined, and a standardized XML serialization of the diagrams that can be detailed to a degree that makes it possible to execute it in an execution engine. By introducing new attributes, the modeler can specify a new intra-WSN pool, containing the logic executed by the WSN. As the latter is resource-constrained, we allow only a subset of BPMN elements in this pool type. Furthermore, we introduced a new WSN activity type. This can be used only within the intra-WSN pool and is (except for the message activity) the only allowed activity type there. As WSNs are inherently distributed systems, we introduced a Target attribute for WSN tasks and sub processes within the intra-WSN participant, that allows specifying where the respective logic should be executed, based on labels that are relevant at the modeling layer. Finally, we added performance annotations, expressing that the WSN should optimize its operation for a specific goal (e.g., system lifetime or reliability) within a certain subsets of activities.

To assist the process modeler in creating correct, executable models, we use a set of meta information that describe the WSN in terms of the logical functionality it provides, along with the way it is embedded into the physical set-up (e.g., which sensing or actuation is supported at which logical location). The most important meta information is which concrete versions of *Tells, Reports, Targets, Data Operators, Local Actions, etc.* are available in a given WSN. This information is currently entered manually, however, later version of `make`*Sense* tools should discover the functionality dynamically.

At run-time, the BPMN process is executed in a distributed fashion. For message exchange between the intra-WSN participant and the other participants, the run-time uses a lightweight protocol, reducing encoded message size by using message structure information on both sides. Communication endpoints caring for serialization and deserialization of messages and for process instance correlation are generated automatically as part of the compilation process.

Figure 3.3 shows the **example ventilation on demand WSN application** modeled as a an extended BPMN diagram. On top and at the bottom, you can see two WSN-aware parts of the process that run outside the WSN. They communicate with the intra-WSN process part in the middle. In that part, you can see all newly introduced BPMN elements. The boxes with dashed lines represent *performance annotation* that tell the system what self-optimization goal to use when the execution is currently in the marked regions. At the beginning of the process, when a message comes in that signalizes a future meeting, the system optimizes for low power consumption until the meeting actually starts. The tasks with the antenna symbol are the *WSN tasks* used for distributed sensing and actuation operations. Sensing operations store their result in data objects (looking like document icons). The decision points, message send tasks, and actuation tasks can read those data objects.

Each WSN task is backed by an attached diagram as shown by the example in Figure 3.4, where the modeler can visually compose the concrete implementations of the abstractions (shown in Figure 3.5) available in a given configuration. By combining concrete *Tells, Reports, Targets, Data Operators, Local Actions, etc.* you can realized local, remote and distributed sensing and actuation with intermediate data processing all on one WSN task.

In order to validate the usefulness of the BPMN modeling approach, we **conducted a user study** that indicated that the modeling tool and the concepts behind WSN application modeling are quite easy to grasp for an audience with some technical background. The concepts that could not be mastered intuitively could nevertheless be learned quickly by the users. The time frame required for teaching basic concepts was comparably low, even as a broad range of topics from WSNs (mainly the make*Sense* abstractions), to BPMN (subset of allowed artifacts and special execution semantics) to instructions on how to use the editor needed to be covered. Given that handbooks and, as in our case, a tutorial are common practice for guiding users in how to use software, we are confident that future users will find using the make*Sense* modeling tool equally easy.

The make*Sense* project also aims at facilitating the changing of processes running in WSNs. In case a process needs to be changed, the current process model is already available to the user as a blueprint and example, which should facilitate the modeling further.

It needs to be noted that given the selection of participants taking part in the study, it is difficult to make generalized statements about how a typical domain expert/business user will be able to handle the tool. However, BPMN was designed to meet the requirements of this target audience. Moreover, we imagine that a domain expert will be supported by a modeling expert, at least in the initial stages of setting up a WSN and for finalizing the executable model. Nevertheless, we recommend that any future user of the modeling tool should have at least a moderate level of technical understanding. Overall, the data collected in the usability study indicates that the developed concepts and tools indeed serve to facilitate the programming of WSNs and make WSN programming accessible to a broad user base. This, in turn will contribute to lowering the total cost of ownership of WSN installations, as well as opening the door for creative uses of WSNs by the domain expert community.

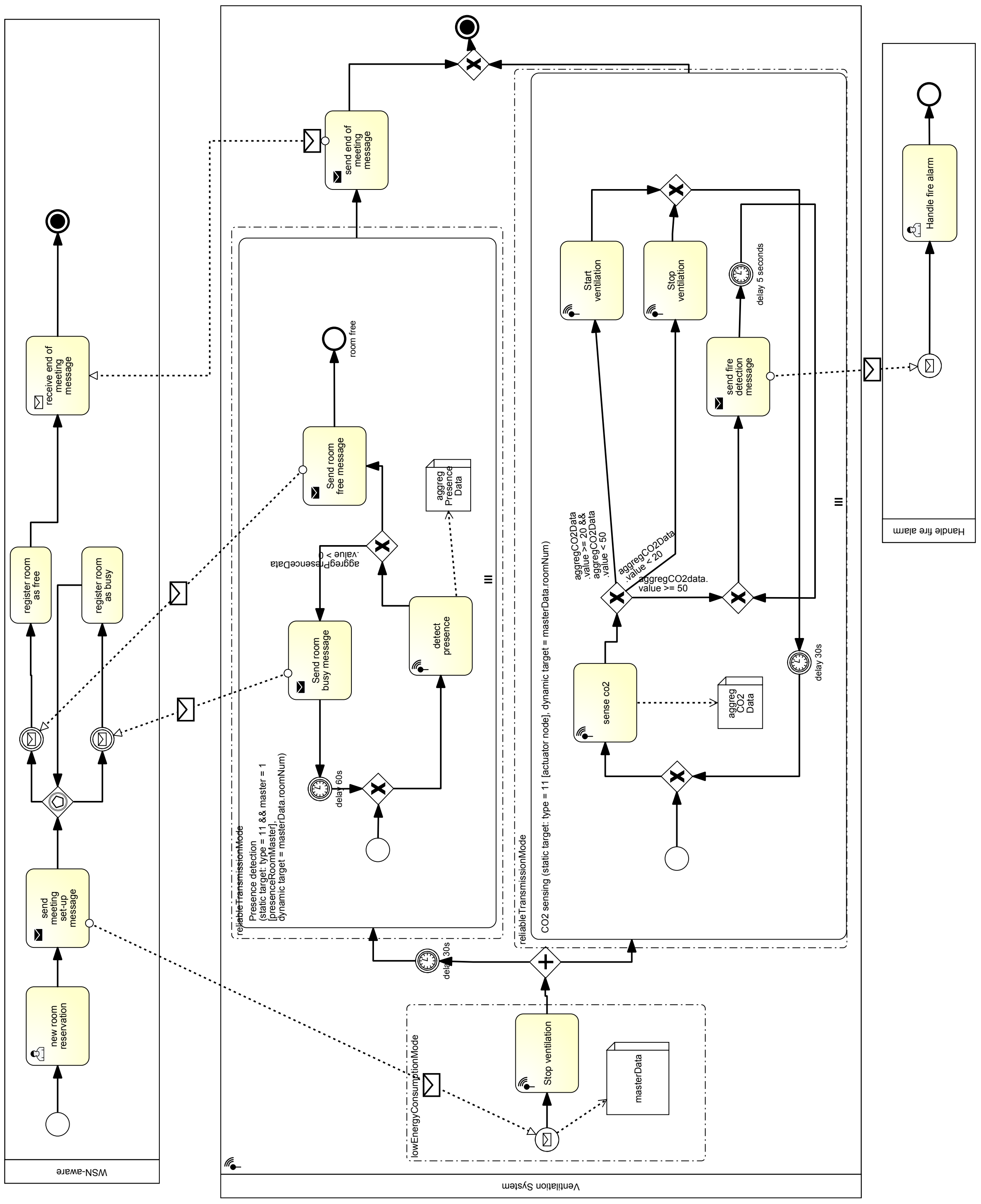

Figure 3.3: The *ventilation on demand* WSN application modeled as a business process.

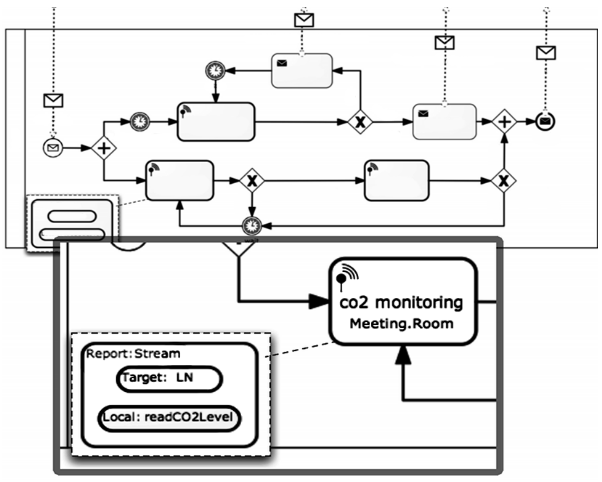

Figure 3.4: Abstraction Composition Diagram

3.4 Macroprogramming

This workpackage aimed to design, implement and evaluate the macroprogramming framework at the core of the make*Sense* approach, including a novel macroprogramming language and a compiler technology enabling the translation of business processes' high level constructs to the single-node run-time functionality.

The **macroprogramming language** is a stripped-down version of Java we tailored for WSNs and is based on a core set of *meta-abstractions* which define the fundamental building blocks of the language as units of functionality, reuse, and extensions. They are implemented through different "concrete" abstractions and provide the key concepts enabling interaction with the WSN. The language serves as the "glue" among abstractions, whose composition can be achieved by using common control flow statements.

Figure 3.5 shows a UML meta-model for the meta-abstractions provided by the macroprogramming language. The core of the model is the notion of *action*, a task executed by one or

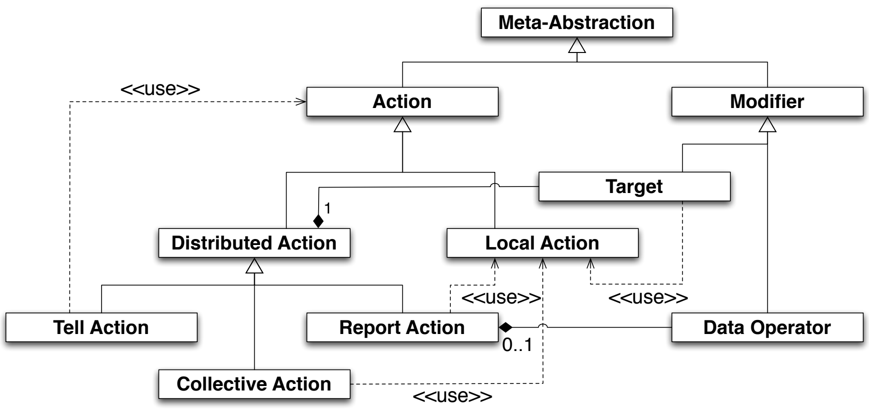

Figure 3.5: A model for the meta-abstractions of the make*Sense* macroprogramming language.

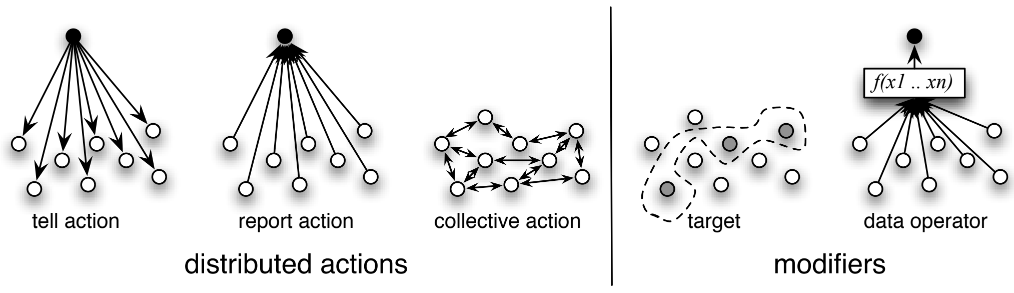

Figure 3.6: Distributed actions and modifiers: a graphic intuition.

more WSN nodes. Actions are separated into *local*, whose effect is limited to the node where the action is invoked (e.g., acquiring a reading from an on-board sensor), and *distributed*, whose effect instead spans multiple nodes. The behavior of distributed actions can be customized by a *modifier* through which programmers are able to map actions to a set of nodes of interest.

Distributed actions are further divided into *tell*, *report*, and *collective* actions. The former two represent the one-to-many and many-to-one interaction patterns commonly used in WSNs to enable communication between the node (the "one") issuing the action and a set of nodes (the "many") where the latter is executed. A tell action enables a node to request the execution of a set of actions on other nodes, while a report action enables a node to gather data from other nodes. Collective actions instead enable a global network behavior and are executed cooperatively by the entire WSN through many-to-many communication. As shown in Figure 3.5, the execution of a tell action depends on a generic (i.e., either local or distributed) action, in contrast to the other distributed actions that instead depend solely on a local action.

Figure 3.6 provides a graphic intuition of the relationship between distributed actions and modifiers, as well as of the meaning of each of them. In the figure, the black node simply represents the node on which a tell (or, respectively, a report) action is invoked, not necessarily a "special" sink node. We defined two modifiers, *target* and *data operator*. 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. Instead a data operator specified on a report action enables the processing of the results after gathering and before they are returned to the caller, e.g., to filter or aggregate the data.

In addition to the meta-abstractions above, we introduced *script actions* as a way to specify an action whose behavior can be an arbitrary code fragment exploiting local actions, distributed actions, or both. Note that this is not to be considered a meta-abstraction, as it is extended directly by the application-level programmer, not the language designer.

General concepts and operations defined by meta-abstractions are implemented by concrete abstractions, which are classes with a predefined interface. As abstraction implementations typically closely interact with the operating system, methods of abstraction classes are implemented in C using a native code interface provided by the core language. The macroprogramming language also provides a special data type to support domain specific languages needed by some abstractions to specify their behavior. Such embedded code snippets are then compiled by appropriate macro-compiler plug-ins, instead of being interpreted at runtime. High level description of application-specific details of the concrete WSN deployment is provided by the *application capability model (ACM)*, already introduced in Section 3.2.2.

The latter can be specified by the application modeler, possibly after consulting with domain experts, and contains a set of application-level attributes and operations made available by the WSN nodes. Example of attributes might be the room where a node is located, or the sensors available to a node.

```
 1  GetTypeLocalAction t = lnew("GetTypeLocalAction");
 2  GetLocationLocalAction l = lnew("GetLocationLocalAction");
 3
 4  code neighborhoodDef = {:
 5    neighborhood template co2Room1() {
 6      t._getType() == "co2"
 7          and l._getLocation() == "room1"
 8    create neighborhood co2Room1 from co2Room1()
 9    }
10  :};
11
12  Target co2Room1 = lnew LN(neighborhoodDef);
13
14  Report co2Stream = lnew Stream();
15
16  co2stream.setTarget(co2Room1);
17  co2Stream.setAction(lnew ReadCO2LevelLocalAction());
18  co2Stream.setDataOperator(lnew MedianOperator());
19  co2Stream.setParameter("period", 60);
20
21  co2Stream.execute();
22
23  while (!co2Stream.isDone() && co2Stream.hasResult()) {
24    Object result = co2Stream.getResult();
25  }
```

Figure 3.7: Use of abstractions in the macroprogramming language.

Figure 3.7 demonstrates the use of abstractions in the macroprogramming language, including the embedded code needed by Logical Neighborhood [6], that is our implementation of the target meta-abstraction. The example shows how to gather readings from sensor nodes of a predefined type (“co2”) and placed in a predefined location (“room1”). Both type and location are application-specific attributes set on the nodes through the ACM. At lines 4 to 10 the target scope is defined by an abstraction-specific code fragment and assigned to a code-type variable `neighborhoodDef`, which is then used to instantiate a target object (line 12). Note the use macro functions getting the value of node attributes through specific local actions (previously instantiated at lines 1-2) and the use of the newly introduced `lnew` operator to create an automatic object instance for which memory management is handled by the macro-compiler. At line 16, the target is assigned to a newly created report action. In the following lines, additional parameters are set, specifically the local action to be executed on target nodes, a median operator used to aggregate the results and the reporting period (60 seconds). Then, the report action is finally executed (line 21). After that, the program waits for the results from target nodes and fetch them as they arrive (lines 23-25).

The macroprogramming language mainly serves as an intermediate language for the translation of BPMN models to platform code, but it is also suitable for direct use by programmers. The core language features a Java-like syntax and full support for object-oriented programming. In addition we decided to provide full multi-threading with a Java-like interface based on the Contiki `mt` library. Since we are targeting very resource constrained devices some standard Java features have been removed. For example, our language does not provide garbage collection, but relies on manual memory management. However it provides specific constructs to allocate automatic or static objects, for which the memory management is handled by the compiler, as well as a generic object serialization interface. The latter is primarily used by the different abstractions in order to transfer object state among nodes involved in a distributed action. In contrast to Java we do not employ a virtual machine approach, but the program is translated to target code that can be directly run on the target platform. The resulting code is predeployed on all nodes, so that it is not necessary to migrate code fragments at run-time.

The **macro-compiler** is responsible for the translation of a macroprogramming language program to Contiki-based C code. The latter can then be compiled with the existing Contiki tool chain and finally deployed on the nodes.

The compilation process consists of four major phases: scanning and parsing, semantic analysis, target code generation, and code partitioning. To support different platforms, like Contiki and TinyOS, it is possible to replace the generation back end, but the currently implementation only supports Contiki.

To reduce the size of the deployed program image, the single macro-program specifying the behavior of the whole network is partitioned into node-specific program parts. Each segment only contains those classes that are potentially executed on the nodes belonging to the respective class. For example, it is not necessary to provision program code for actuator control on pure sensor nodes. In the current implementation, we only differentiate between regular nodes and a dedicated gateway, but this concept can be easily extended to a larger number of node classes.

To support the embedded code fragments, the macro-compiler exhibits a plug-in interface that allows to integrate small sub-compilers for the abstraction-specific languages. Each of these plug-ins is responsible for parsing, type checking, and translation of the respective code fragments. The plug-ins are automatically invoked by the main compiler, if it encounters an embedded code fragment in the macroprogramming code. A return channel allows the plug-ins to inform the compiler about references to macroprogramming language constructs encountered in the embedded code fragments. Like the macro-compiler, the plug-ins are implemented in Java.

An **evaluation** of the macroprogramming model has been carried out to study how programmers handle the make*Sense* macroprogramming framework and also to assess the performance of the resulting code in comparison to implementations in native Contiki. In the first case we performed a user study with novice sensor network programmers, the main target of the make*Sense* framework. The overall results, summarized in Figures 2.3 and 2.4, let us claim that even though some additional implementation work is needed to further improve specific parts of the system, the macroprogramming model successfully achieves the project's objectives, namely, simplifying the programming of integrated sensor network applications.

To evaluate the performance of the make*Sense* programming model and the make*Sense* programming language we selected a small set of WSN applications and implemented each as an MPL program as well as a classical Contiki-based WSN program. We compare the performance of both implementations based on a set of typical software performance metrics, like code size, memory consumption, and number of sent messages. The results indicate that even though the make*Sense* macroprogramming framework introduces an overhead in the evaluated example applications, that is small enough to support resource contained devices. On the other side,

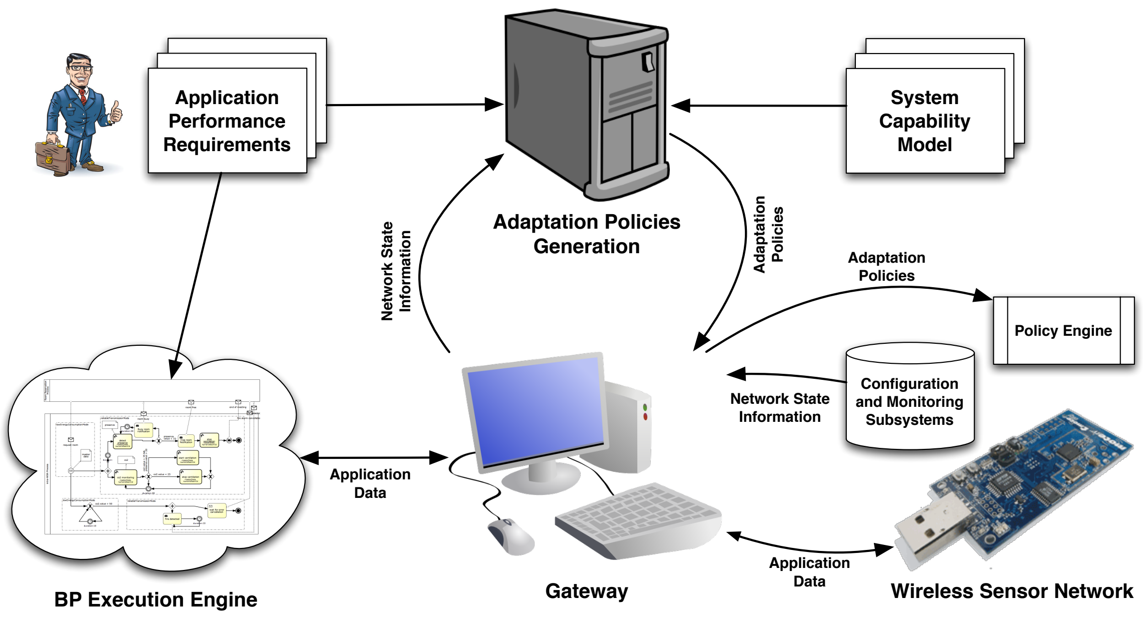
Application Performance Requirements
System Capability Model
Adaptation Policies Generation
Network State Information
Adaptation Policies
Adaptation Policies
Policy Engine
Configuration and Monitoring Subsystems
Network State Information
Application Data
Application Data
BP Execution Engine
Gateway
Wireless Sensor Network

Figure 3.8: make*Sense* run-time architecture.

Contiki-based implementations are strictly tied to considered task and network structure. An implementation providing a similar level of flexibility and features would be far more complex and consequently even more difficult to implement and maintain. Consequently, we can state that the overhead for supporting object orientation and powerful abstractions is still reasonable to be also suitable for resource constrained devices, like sensor nodes.

3.5 Run-time System

The make*Sense* run-time system support the execution of make*Sense* applications on resource-contrained sensor nodes and their dynamic reconfiguration. Figure 3.8 shows its high-level architecture. The business process execution engine connects to the sensor network through a dedicated gateway we design. Application performance requirements are specified in the extended business processes. These are taken as input by a dedicated optimization engine that generates *self-optimization* policies that allows the network to dynamically tune its behavior. The latter task is carried out based on information from the system capability model and network state information from the deployed network. On the sensor nodes we deploy a dedicated *configuration and monitoring* subsystem that oversees the application execution inside the sensor network and executes the adaptation policies depending on the observed state. Orthogonal to the functionality running when the main application is in operation, make*Sense* also supports dynamic reconfiguration of WSN binary code through a dedicated *selective reprogramming* facility.

While the make*Sense* gateway is implemented with mainstream technology as it is intended to run on a standard machine, the key functionality of the make*Sense* run-time system lies within the configuration and monitoring subsystem aboard the sensor nodes, in the generation of self-optimization policies, and in the support to selective reprogramming. We describe these components next.

3.5.1 Monitoring and Configuration

The key design principle of the configuration and monitoring subsystem is to separate protocol logic from configuration [7]. This way, parameters in all parts of the system can be configured through a separate configuration component based on the settings that the self-optimization policies dictate. This makes it simple to handle changes in the objectives of the application, e.g., when the application demands a new objective such as high throughput instead of low energy consumption. Furthermore, we aimed at keeping a layered design to make it possible to exchange layers, for example, when a new MAC layer should be used. While researchers have argued that cross-layering is required in WSNs to achieve high performance, in previous work we showed that we can both rely on a layered system and achieve high throughput [7].

In designing the configuration and monitoring functionality, we aimed at lessening the burden on developers of configuration policies due to gathering and processing the data input to the self-optimization mechanism. To this end, we opt for a unified tuple space-like API spanning both read and write operations on the local blackboard, and distributed operations to share the configuration and monitoring information across 1-hop neighboring devices [8]. We also aimed at a design that has clearer boundaries and hence requires little re-engineering work when new Contiki releases are available. Therefore, we use wrappers between Contiki components, e.g., the MAC protocol, and our configuration run-time.

As shown in Figure 3.9, the configuration and monitoring subsystem includes a central blackboard for storage of configuration parameters, system state, and statistics. The other modules access the blackboard storage via tuple space-like APIs [8] that operate on the relevant data. make*Sense* modules handle their configuration directly via the blackboard while non-make*Sense* modules, such as Contiki components, are wrapped so that relevant configuration and state can be stored in the blackboard. The monitoring modules are responsible for acquiring information on performance and resource consumption, storing it in the blackboard to make it available to upper layers.

The configuration policy and policy engine are responsible for setting the performance-related parameters. They also provide the interface to the optimizer that runs outside the network, as described next. The policy engine enforces the relevant policies by setting appropriate parameters in the blackboard that determine the corresponding modules' behavior and performance. As any initial configuration is likely to be sub-optimal, the optimizer will dynamically update the configuration over time.

3.5.2 Self-optimization

In several real-world deployments the application and operating system code are finely-tuned to achieve a certain performance goal [9]. Most often, this is based on the developers' intimate knowledge of the internal sensor networks mechanisms and a deep understanding of the application requirements. The deployed code is also entirely in the hands of the same developers, who are free to tune the implementations depending on the performance goals.

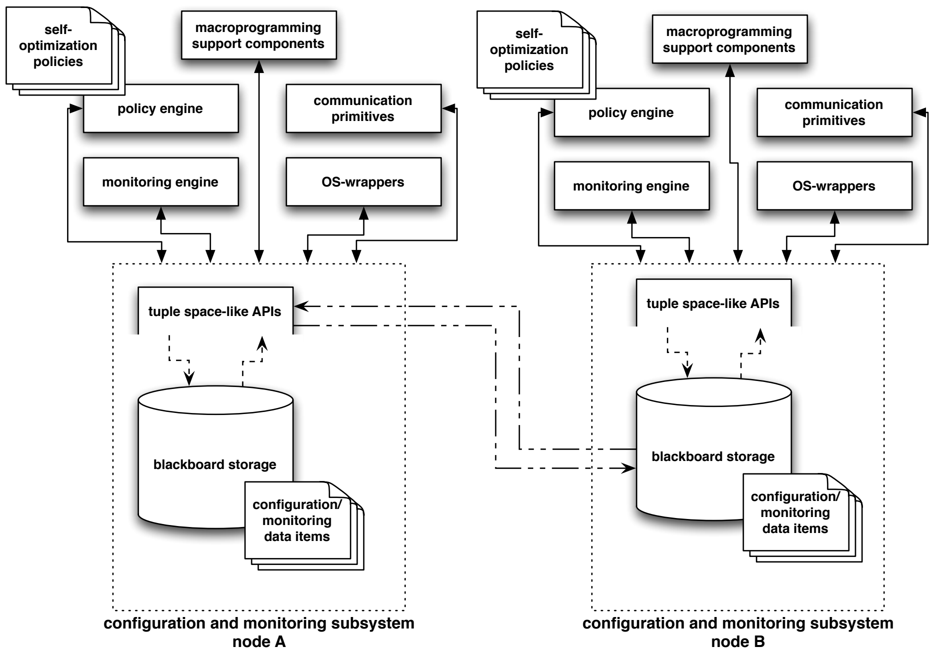
self-optimization policies
macroprogramming support components
policy engine
communication primitives
monitoring engine
OS-wrappers
tuple space-like APIs
blackboard storage
configuration/ monitoring data items
configuration and monitoring subsystem node A
configuration and monitoring subsystem node B

Figure 3.9: Overview of the configuration and monitoring subsystem

In general, the approach above is not possible in make*Sense*. Two main reasons concur to this: *i)* the executable code is generated from high-level application models, and the mapping from the latter to low-level Contiki C is not trivial; and *ii)* the programming framework is open to external developers, who may contribute new concrete abstractions along with their supporting run-time. Furthermore, make*Sense* allows application developers to specify performance objectives that can change at the run-time. This is necessary to support long-lasting business processes subject to rapidly changing requirements. Therefore, the make*Sense* run-time must be able to *self-optimize* towards the stated performance objectives.

We define *self-optimization* as the property of a system to automatically find near-optimal system configurations whenever application objectives, system parameters, or environmental conditions change. To enable self-optimization, we gather run-time information from the deployed sensor network, e.g., network topology and protocol performance, and feed these to a reinforcement learning algorithm that explores the space of possible configurations using simulations.

The learning is performed using a plug-in for the Cooja simulator. A utility function based on the performance objectives provides the reinforcement learning with the needed rewards to implement the learning process. The specific learning mechanism that we use is First Visit Monte Carlo Policy Iteration [10]. We use the Cooja simulator as it allows to to accurately emulate sensor nodes such as TMote Sky and Wismote. This makes it possible to reuse the firmwares that are executed on the real sensor network in the simulator, making the simulation behavior as realistic as possible.

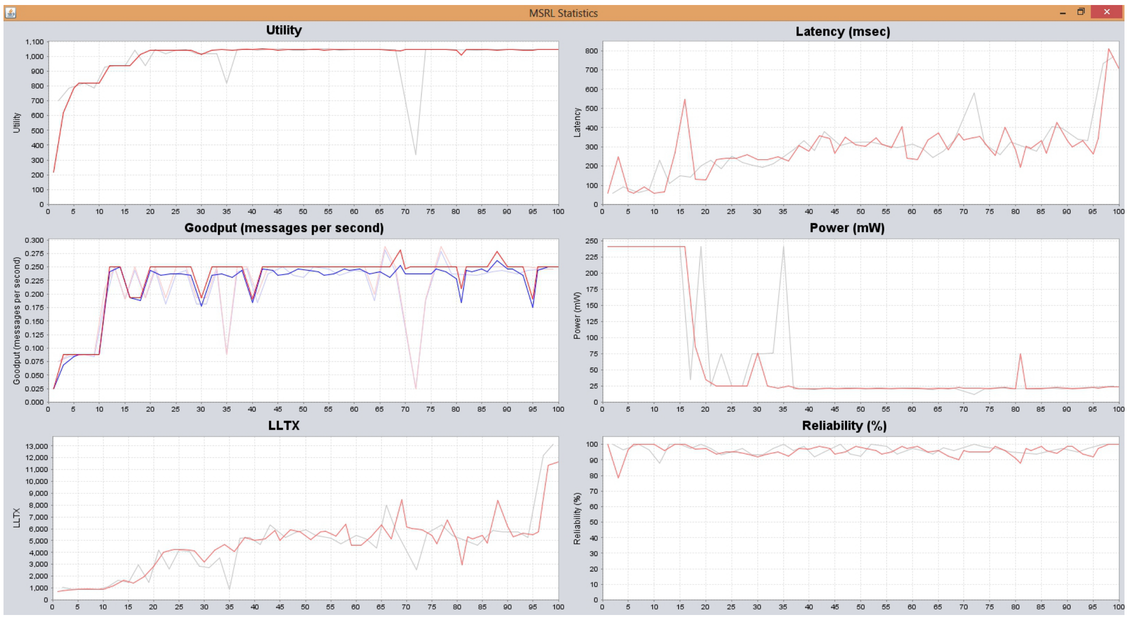

Figure 3.10: Applying self-optimization in the ventilation on-demand scenario.

At the end of each simulation round, the learning process evaluates the performance obtained with a given setting w.r.t. the application's performance goals. Based on this, we derive self-optimization policies that specify which parameters provide better performance as a function of the current application and environment state. In its simplest form, a policy is a mapping between a state and a set of actions that should be performed when the application is in this state. An action in this case can be a value to update in the blackboard that triggers a reconfiguration. We distribute the policies back to the deployed network where nodes will apply them whenever needed.

This approach sharply differentiates from existing solutions. Rather than requiring detailed modeling of the individual protocols, as done for example with great effort for MAC protocols [11], we treat the entire application as a black-box. This may lead to sub-optimal solutions, but also enjoys greater flexibility as it lets users add programming abstractions to the framework along with their supporting protocols and have the latter "implicitly" optimized.

The results we obtained confirm the effectiveness of our approach. For example, in the ventilation scenario we assume a performance objective that optimizes for long lifetime of the network and enough throughput for the ventilation control system to run its control loop. Figure 3.10 shows the result of running 100 learning episodes in the ventilation scenario. In this experiment we used a network with four sensor nodes and one sink node. One node has two neighbors, the rest has three. The simulation corresponds to a room with four sensors installed with a small variation in density. In Figure 3.10 the utility improves significantly over time: the learning algorithm first detects that it is better to send many messages (favoring goodput), and after a while also turns on duty-cycling (seen in the energy graph where the energy falls between episode 15 and 20).

`disseminate(target, image, pages, toAck)`	requests nodes in `target` to disseminate the selected `pages` of `image`
`receive(target, image1, pages1, image2, pages2, toAck)`	requests nodes in `target` to receive the selected `pages` of `image`; optionally, these are loaded along with the selected pages of `image2`
`idle(target,toAck)`	forces the nodes in `target` to return to `IDLE` state
`switch(target, image, toAck)`	requests the nodes in `target` to reboot and load `image`
`query(target)`	retrieves from the nodes in `target` a description of system information
`send(target, message)`	sends `message` to the nodes in `target`, in multi-hop if necessary

Table 3.1: CodeLeash API.

3.5.3 Selective Reprogramming

make*Sense* applications require maintenance, just like any other software artifact, due to bugs or to users demanding updates or new features. Sometimes the functionality of the reconfiguration and monitoring system is not enough because some scenarios may require a reconfiguration of WSN binary code through a reprogramming service, according to run-time changes in high level objectives specified by the user.

In designing the selective reprogramming tool, named CodeLeash, we aimed to move away from the application-agnostic, single image, network-wide reprogramming by providing the user with an expressive API to define application-specific reprogramming policies over specified target nodes. As a consequence, the tool has been conceived to support both scenarios where the reprogramming policies are directly controlled by a human operator, and semi- and fully-automated scenarios where the policies are executed directly by the nodes. Moreover, we aimed at providing a sharp decoupling of the reprogramming service and the main application in separate code images swapped at appropriate times, thus minimizing the potential interference between them. As a consequence, the user can control *who* should receive an updated image, *what* they should receive, and *when*. For instance, in the make*Sense* *ventilation on demand* scenario one could reprogram one room at a time, moving to the next only after verifying that the current one is operating correctly or (re-)deploy functionality with a particular role (e.g., an aggregator) only on nodes with enough energy and good connectivity.

As such, CodeLeash is a single tool providing a rich set of alternatives for WSN reprogramming w.r.t.: *automation* provided to the operator, *distribution* inside the network and *application-awareness* of the dissemination policy used. To support those degrees of flexibility we expose the fundamental building blocks of the system, the *core abstractions*, through a common API used by developers according to their dissemination strategies. Core abstractions provide the key concepts enabling interaction with the reprogramming tool and are associated to specific operations (*commands*) that nodes must execute according to a specific dissemination policy. A core abstraction can be a command itself (e.g. tell a node to disseminate an image or to get ready for receiving an image) or a command parameter (e.g. to specify the target nodes of a command).

CodeLeash API is shown in Table 3.1. All the commands accept a `target` parameter which denotes the set of nodes on which the command should be performed. In our current

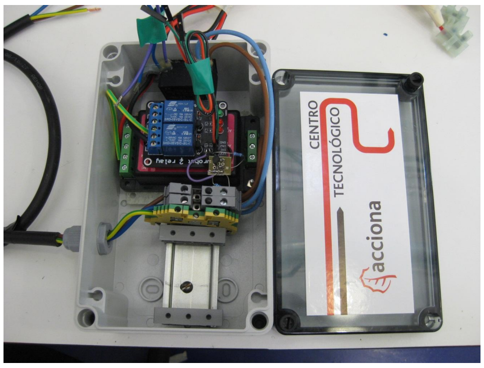

Figure 3.11: Actuator connection

implementation, the target is composed by a pair ⟨nodes, hops⟩ where the former is a set of node identifiers (or the `ALL` special keyword) and the second is the "scope" within which the nodes should be found, specified in number of hops from the node issuing the command. Further, a binary image is identified by a tuple ⟨id, version⟩; the `pages` parameter is a bitmask selecting the set of pages of interest within the image. Finally, the flag `toAck`, if true, requests that the outcome of the corresponding command, after completion, is communicated back to the caller.

The core of the system is constituted by two different configurations used at different times. During normal operation, the make*Sense* application loaded into memory is linked to a small component, namely the *sentry*, which intercepts incoming messages and serves those containing reprogramming commands. Interestingly, the sentry serves directly only `switch`, `send` and `query` commands; the other commands are instead provided by a second configuration, namely the *reprogrammer*, whose loading into memory is triggered by an appropriate switch command. The reprogrammer implements the full API and executes the behavior in the specific reprogramming policy. This design minimizes the amount of memory consumed by the tool during normal operation, and at the same time allows for a reprogramming functionality of arbitrary complexity, which in principle can use the entire code memory available. Application image switching is allowed by a custom bootloader, developed along with the tool, that is able to load images from different partitions of the external flash of sensor nodes.

3.6 Real-world Deployment

We have implemented a variation of the ventilation-on-demand scenario described in Section 3.2.1, and deployed it in a student residence in Cadiz, Spain. These rooms were equipped with CO_2 sensors in a previous EU project that we could reuse. In every student room, the CO_2 sensor would possibly operate an actuator controlling a flap. This opened and closed depending on the measured CO_2 value. Figure 3.11 and 3.12 show the devices used in the installation.

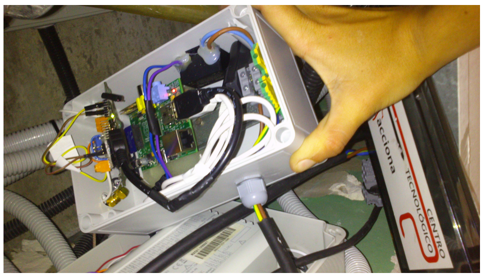
Figure 3.12: Node attached to the CO_2 sensor

3.6.1 Implementation

The actual implementation of the deployment in Cadiz was performed using the `make`*Sense* macroprogramming language. An additional feature is that due to the noise that the actuator made, the flap was not to be operated during night time. The latter functionality could be easily accommodated using a `Local Action` that was running on the gateway to gather the local time and send a command (using a `Tell` concrete abstraction) to the master nodes telling them to stop or to resume operation. The implementation also included advanced features such as a `ScriptAction` to decentralize the operation of the control loop.

Although this implementation misses the actual integration to an external business process due to the strict scheduling of the operation in Cadiz compared to the testing of the model compiler, the actual connection between the sensor network processing and an external business process engine was successfully tested separately.

3.6.2 Functioning

The system deployed in Cadiz has been up and running for about a week. We assessed its functioning through the logs collected at the gateway and by co-located RaspberryPI devices whose only task was to dump on local storage the serial line output of the installed nodes.

Based on the information in the logs, we confirm that the system worked correctly for the whole duration of the deployment. This included the shutdown operation at night we implemented not to disturb the sleeping of the students with the noise generated by actuators. During day time, the actuators operated the attached fan twice every hour, on average. Nevertheless, we can identify two specific behaviors. When a person is in a room, the CO_2 readings slowly increased until the threshold we set was passed. This triggered the actuator to open the fan, causing a decrease in CO_2 readings until the values were back below the threshold. At that point, the actuator closed the fan and the behavior started to repeat as long as a person was in the room. On average, such periodic behavior happened every 15 minutes. On the other hand, it is evident from the logs when a person was not in a room, because of the constant and below threshold CO_2 readings.

3.7 Industrial Business Evaluation

We have performed a life cycle cost analysis comparing the total cost of ownership of a building automation system for (1) a conventional wired system with (2) a WSN installation and (3) a WSN installation that uses the make*Sense* approach. To estimate the costs, we have used the knowledge gained through the experience of ACCIONA from three previous research projects, namely, *Clear-Up*, *Fiemser* and *Arrowhead*. For the make*Sense* case and the comparison with the Contiki/C case (the hardware would be the same), we used both ACCIONA's experience from previous WSN deployments backed up with expertise of the consortium and the experience from our own make*Sense* deployment.

The estimations are based on a deployment with 30 nodes since the consortium has experience with deployments of that size. We also assume that the cost of the marginal components increases linearly with the number of nodes. Regarding the cost estimation of the make*Sense* system, we have been conservative w.r.t. the cost savings. In a real world deployment, the savings could be even higher. The cost estimations are for a deployment that is operated for five years before it is disposed.

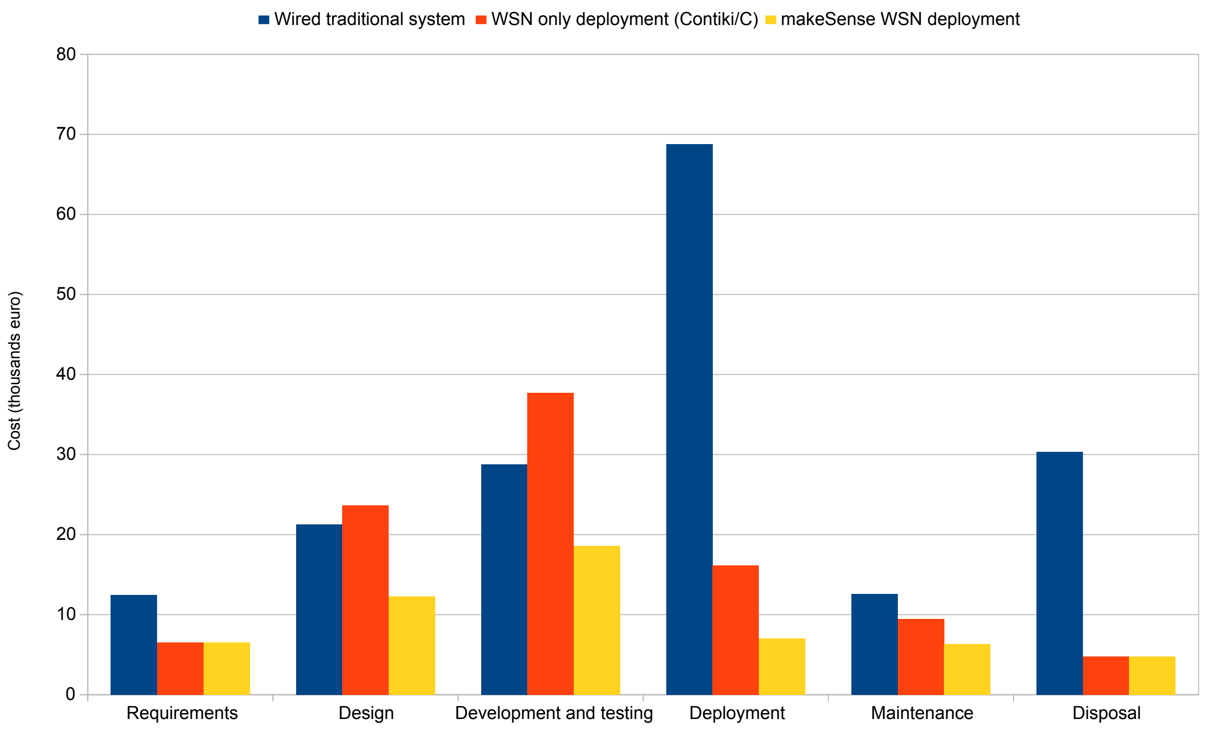

Figure 3.13: Life cycle cost categories aggregated, costs in thousands of euro. make*Sense* reduces costs drastically, in particular during design, development & testing as well as deployment.

Figure 3.13 shows the aggregated costs for *(i)* requirements analysis, *(ii)* design, *(iii)* development and testing, *(iv)* deployment, *(v)* maintenance, and *(vi)* disposal. Figure 3.14 shows the total cost. The total cost would be about 180.000 Euro for a wired system. A conventional wireless sensor network system would lower the costs already to 105.000 Euro

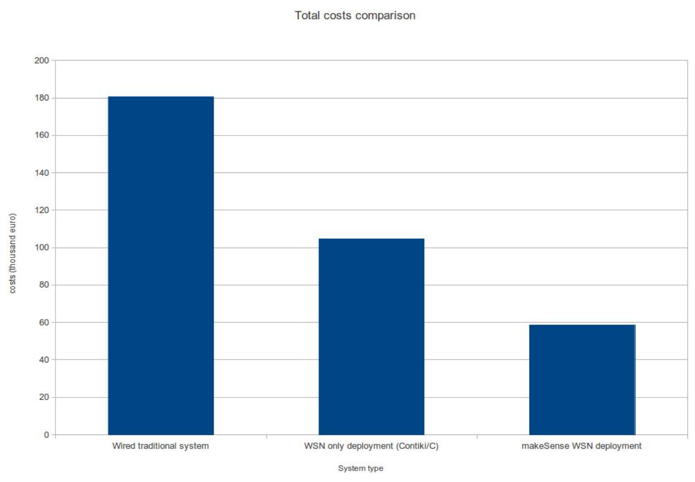

Figure 3.14: Life cycle total cost for a 30 nodes deployment. make*Sense* reduces the cost to less than 60.000 Euro compared to 105.000 Euro for a conventional wireless sensor network and 180.000 Euro for the equivalent wired system.

while a system developed with the make*Sense* approach would be disposable for a mere 60.000 Euro. Compared to a wired system, the major cost savings are, as expected, during deployment and disposal since the cost of associated with wiring and removal of the wires is avoided. Compared to a conventional WSN, the major cost savings for make*Sense* are in the design, development and testing phases. The savings stem from the fact that design and development with the graphical editor are simpler and the auto-generated code contains less bugs than its hand-written counterpart. The latter reduces the costs for testing.

3.8 Project Dissemination and Tutorial

In this section, we present the dissemination activities. We start with the tutorial since it was one of our major vehicles for promoting the make*Sense* results.

3.8.1 Tutorial

The make*Sense* **tutorial** has the form of a downloadable virtual machine with a tutorial document. It has been actively used as a dissemination tool and that has made a significant number of researchers and practitioners use the make*Sense* software.

The tutorial has two parts. Part I is for domain experts with very limited or no knowledge on sensor networks. It describes how to use our business process model editor to develop wireless sensor network applications using the graphical editor with an extended version of BPMN, the Business Process Modeling Notation, standard. Part II is for people with programming experience (but not necessarily sensor network experts) and describes the make*Sense* macro-programming language and shows how to develop executable applications that can be executed in the simulator or on real hardware.

In Part I, the tutorial users are guided through a process where they create a WSN-enabled business process, that senses some environmental conditions and controls the forced ventilation in the meeting rooms of a company. The goals are to save energy by only ventilating the room when it is booked and occupied by employees and to more accurately monitor and charge room occupancy. This is a step-to-step introduction where the tutorial users first are shown how to define the control flow and message exchange in the WSN using a diagram in BPMN. Then they learn how to define WSN actions that describe how sensor data is collected and actuators are driven.

As mentioned above Part II is for programmers. As all tutorials on programming, it starts with the “Hello World” program. The tutorial introduces the different meta-abstractions such as `target` and its *Logical Neighborhoods* implementation. After this basic introduction, actuation is introduced with the “Blink” program that turns a node’s LED(s) on and off at fixed time intervals. To implement this functionality, two make*Sense* components are needed: a `LocalAction` for turning an LED on or off `SimpleTell` that tells the nodes to execute this `LocalAction`. In the following section, `ScriptActions`, a very powerful abstractions in make*Sense* are introduced. They allow one to encapsulate a sequence of actions and send it to a specified target for remote execution. The users rewrite the “blink” application and can see that this solution produces less network traffic, which saves energy. After the introduction of a sensing application, users are shown how to write a complete application containing both sensing and actuation.

Both tutorial parts also include exercises the tutorial users may try.

3.8.2 Video

In order to disseminate the make*Sense* approach to a broad audience, we have produced a **high quality marketing video** that demonstrates the main ideas of make*Sense* and that has caught a lot of interest and received around 1700 views[1].

The video first introduces wireless sensor networks with some application examples noting that this can drive business processes. Then end-user ACCIONA and their business areas are introduced. The video continues stating that ACCIONA wants to apply sensor networks, for example, for ventilation of rooms but that they are difficult to program which makes them expensive. Furthermore, it is almost impossible to find someone who knows both the application domain and sensor networks. This introduction lays the motivation for the make*Sense* project. After the explanation of this background, a senior person from SAP explains the advantages of make*Sense* and SAP lead Patrik Spiess names the partners and what they bring into the project.

From the end-user ACCIONA the connection to the deployment at Cadiz is made, the deployment and its goal of the reduction of energy for ventilation of student dorms is introduced pointing at the high possible savings. Next, the advantages of the integration with the business processes are exemplified. Finally, again the advantages of make*Sense*, i.e., empowering domain experts to develop integrated sensor network application is highlighted leading to reduced cost of ownership of such integrations.

[1]The marketing video is accessible over the make*Sense* website or directly at `https://www.youtube.com/watch?v=n4ospuvPrWA`

3.8.3 Website

Already in an early stage of the project, we established the project website at the address `http://www.project-makesense.eu` and constantly updated it with the latest developments. The use of the .eu top-level domain strongly links the project to the European Union and the Commission as the project's co-founder.

3.8.4 Publications

In make*Sense* we achieved the acceptance of two publications at top-tier conferences with an acceptance rate below 20%, namely the paper at the NIER (new ideas and emerging results) track at the **International Conference on Software Engineering (ICSE)** and the paper at the **International Conference on Business Process Management (BPM).**

To ensure sharply focused communication to the scientific and industrial community who is likely most interested in the make*Sense* results, a **final make*Sense* workshop** was organized as a special session in the 4th International Workshop on Networks of Cooperating Objects for Smart Cities 2013 (CONET/UBICITEC 2013). We strategically chose this setting as it allowed us to disseminate to both a European audience (many participants were from the EU) but also making the project known to the international community. The workshop was held at CPSWeek in April 2013 in Philadelphia (USA) and attracted about 20 participants.

Other than this, the project participants used every opportunity to promote the project through invited talks, at customer meetings, and in personal encounters at various internal and external events.

In the following, we list just a selection of our mayor scientific publications:

Publications in Scientific Journals

- Luca Mottola and Gian Pietro Picco, **"Middleware for Wireless Sensor Networks: An Outlook"**, Journal of Internet Services and Applications, May 2012.
- Prasant Misra, Luca Mottola, Shahid Raza, Simon Duquennoy, Nicolas Tsiftes, Joel Höglund, and Thiemo Voigt **“Supporting CPS with Wireless Sensor Networks: An Outlook of Software and Services”** Journal of the Indian Institute of Science, Sept. 2013.
- Stefan Guna, Luca Mottola, Gian Pietro Picco, **“DICE: Monitoring Global Invariants with Wireless Sensor Networks”**. Accepted for Transaction on Sensor Networks, expected publication date is November 2014.

Publications in Scientific Conferences and Workshops During the course of the project, the following scientific, peer-reviewed papers have been published:

- Fabio Casati, Florian Daniel, Adam Dunkels, Stamatis Karnouskos, Patricio Moreno Montero, Luca Mottola, Felix Jonathan Opperman, Gian Pietro Picco, Kay Römer, Patrik Spieß, Stefano Tranquillini, Paolo Valleri, and Thiemo Voigt, **“Poster Abstract: makeSense: Easy Programming of Integrated Wireless Sensor Networks”,** EWSN 2011, Bonn, Germany, Feb. 2011.

- 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, **"Demo Abstract: From Business Process Specifications to Sensor Network Deployments"**, EWSN 2012, Trento, Italy, Feb. 2012.

- Fabio Casati, Florian Daniel, Guenadi Dantchev, Joakim Eriksson, Niclas Finne, Stamatis Karnouskos, Paulo Moreno Montero, Luca Mottola, Felix Oppermann, Gian Pietro Picco, Antonio Quartulli, Kay Roemer, Patrik Spiess, Stefano Tranquillini, and Thiemo Voigt. **"Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks"**, Int. Conference on Software Engineering (ICSE)- New Ideas and Emerging Results Track, Zurich, Switzerland, June 2012. *Acceptance rate: 17%.*

- Stefano Tranquillini, Patrik Spiess, Florian Daniel, Stamatis Karnouskos, Fabio Casati, Nina Örtel, Luca Mottola, Felix Oppermann, Gian Pietro Picco, Kay Roemer and Thiemo Voigt. **"Process-based design and integration of wireless sensor network applications",** 10th Conference on Business Process Management, Tallinn, Estonia, Sep. 2012. *Acceptance rate: 15%.*

- Florian Daniel, Joakim Eriksson, Niclas Finne, Harald Fuchs, Andrea Gaglione, Stamatis Karnouskos, Patricio Moreno Montero, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, Kay Römer, Patrik Spieß, Stefano Tranquillini, Thiemo Voigt, **"makeSense: Real-world Business Processes through Wireless Sensor Networks"**, 4th International Workshop on Networks of Cooperating Objects for Smart Cities 2013 (CONET/UBICITEC 2013), held in conjunction with CPSWeek Philadelphia, USA, April 2013

- C. Timurhan Sungur, Patrik Spiess, Nina Oertel, Oliver Kopp, **"Extending BPMN for Wireless Sensor Networks",** 15th IEEE Conference on Business Informatics (CBI 2013), Vienna, July 2013. *Acceptance rate: 25%.*

4 Consumption, Cost and Benefits

4.1 The makeSense product offering

As stated in previous chapters, the make*Sense* tool chain can be applied to any scenario where a technical or business process benefits from sensor information and/or would automatically trigger actuator operations.

The results of make*Sense* can be consumed in two versions:

	Functiality and Benefits	**Cost**
Free version	The free version consists of all academic, open-source tools that were created within the project, comprising • Macro compiler • Programming abstractions and implementation of example concrete abstractions • Sensor run-time for compiled macro program • Run-time optimization tools for optimized performance This enables interested customers to • Program scenarios in macro code • Compile and deploy the application on commodity off-the shelf Contiki sensor nodes	The free version has a simple cost structure: • No up-front or recurring license costs: all software is freely available. • Fixed hardware cost for central components: gateway and application servers. • Linear cost to number or deployed sensor nodes. • Reduced training and implementation cost – Potentially some training cost for the macro programming language and the use of the make*Sense* tools, approx. 50% lower than for conventional Contiki programming. – Development cost for the application itself, approx. 50% lower than for conventional Contiki programming.
Enterprise version	This version includes the free version plus the commercial tools created by SAP, comprising • WSN-enabled BPMN modeler with extended export functionality, containing WSN modeling artifacts • Model compiler which is able to compile the exported extended BPMN model into macro code • Run-time visualization to show the execution status of the currently running process. This enables interested customers to • Model a WSN application with back-end integration without programming • Visualize process execution at run-time	The full version incurs license cost, but lets customers save on development effort, realizing net savings in cost. • The software will be licensed with an individually negotiated license cost, incurring up-front payments. • The customer can choose to by maintenance, including bug fix releases and implementation of requested features. • Reduced training and implementation cost – BPMN modeling knowledge often available within potential customer organizations, approx. 80% less training cost. – Higher speed and less effort required for initial implementation, approx. 80% less cost. – Quick and easy change of modeled process.

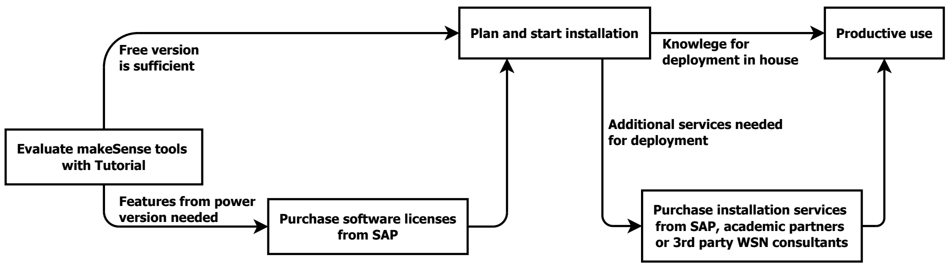
Evaluate makeSense tools with Tutorial
Free version is sufficient
Features from power version needed
Purchase software licenses from SAP
Plan and start installation
Knowlege for deployment in house
Productive use
Additional services needed for deployment
Purchase installation services from SAP, academic partners or 3rd party WSN consultants

Figure 4.1: The Consumption Decision Path

4.2 The Consumption Decision Path

Interested end users can consume the make*Sense* results. The advice is to follow the following consumption decision cycle, as shown in Figure 4.1.

Users can try out the tutorial, where they find a pre-configured, simulated evaluation environment. Here, they can assess the effort required for implementing a WSN application with back-end integration by macro code or with modeling. If the decision goes into the direction of implementing with macro code, they could ask the academic partners or 3rd party consultancies for support. If more extensive support would be required, that could be offered as a payed service. If the end user decides to use the enterprise version of the stack, an inquiry to SAP must be made. License cost will be negotiated on a per-case bases, depending on the number of developers, the number of WSN nodes and the expected saving that the user would realize. Also in this case, the end user might require additional training or installation and configuration support, which could also be purchased by the respective parties.

4.3 Competitive Position

To our knowledge, there is no competitive offering, that combines all aspects of make*Sense*. The ability to define processes in a way that data processing and communication is specified (and not programmed) in the business process description but the execution is completely outsourced to sensor nodes and the ad-hoc network they form is unique.

Indirect competition is present in the form of alternative approaches, that all stream the data out of the WSN part and process it outside of it. This includes *event stream processing (ESP)* products from various software vendors. However, none of these approaches allows to configure the front-end itself.

The unique feature of the make*Sense* tool chain is its ability to define and implement control loops that run completely in the WSN front-end, independent of central back-end systems. This enables scenarios where the sensors and actuators are only intermittently or unreliably connected to the Internet (e.g. in trucks or other logistics services that pass rural areas) or back-end systems or where this uplink is costly (e.g. because it relies on satellite connections like in marine freight scenarios). With make*Sense*, organizations can implement processes that in part run locally in the ad hoc network that the WSN sensors and actuators form automatically and further can synchronize with the back-end, once that connection is available or affordable again.

5 The potential Impact

5.1 Research Impact

make*Sense* has advanced the state of the art in the area of wireless sensor networks and business process management. The key research impacts include:

- make*Sense* integrates business back-ends with wireless sensor networks through process modeling familiar to domain experts
- make*Sense* provides a unified extensible programming framework that will simplify the development of non-trivial wireless sensor network applications either graphically or through a high-level macroprogramming language

Hence, the make*Sense* framework enables other researchers to rapidly prototype and evaluate new ideas. Furthermore, make*Sense* helps researchers to explore new application domains. In particular the development of new programming abstractions will be simplified, since these can be easily tested by integrating them into make*Sense*. Moreover, the make*Sense* framework will make it possible to explore the interactions of new programming abstractions with existing programming abstractions which is currently very cumbersome. Due to the modular, layered approach of make*Sense*, other researchers can exploit new ideas at different levels. New mechanisms can be implemented on top of any layer.

5.2 Impact on Industry

Before we look at impact on a broader perspective, SAP can report some direct impact on products that will be widely sold in the industry. Concepts from make*Sense* have been incorporated in *SAP HANA Platform, IoT Edition*[1]. This new product is targeted at integrating devices from the Internet of Things with SAP systems. Inspired by make*Sense*, the integration between the intelligent *things* in the IoT and SAP back-end systems is modeled rather than programmed.

Additionally, SAP has patented its part of the make*Sense* approach, i.e. application modeling and compilation. This allows SAP to generate value from the IPR created in make*Sense* by commercially licensing this know how to interested commercial users. Other than that, SAP could also give out free licenses, e.g. if some academic or non-profit parties would like to use it.

[1]http://www.sap.com/iot

5.2.1 Addressable market

Several sources assess a huge market potential for WSNs in the US and Europe. Estimations range from approx. \$2 billion by 2022 to 4 billion by 2016 for WSN in the US alone.[2][3]. The world market is estimated at 14.6 billion by 2019.[4] Growth rates are estimated to be between 55 % and 130 % from 2012-2016.[5] A report from Machina Research even estimates the whole impact by 2020 on people and businesses stemming from the sale of connected devices and services to \$4.5 trillion [12]. For a further recent, in-depth report of the potential of WSNs, refer to Karnouskos et al. [13].

As laid out in 4.3, make*Sense* can find its niche in this market, as it is the only framework that comprehensibly supports partly autonomous scenarios where the uplink is a problem. If only 1 % of the applications subsumed in the above estimations entail such deployments, make*Sense* can generate significant value by addressing this niche. But as shown before, a broad range of other scenarios can benefit from the free version of the make*Sense* tool chain, without any additional cost for the end user incurred.

5.2.2 Benefits at Organization Scale

If companies are to invest in the make*Sense* approach, one of the major questions is how they can benefit from it, and more concretely how make*Sense* can help them to save money. In general, make*Sense* will (as our life cycle cost analysis has shown) reduce costs by requiring less effort for WSN setup, maintenance, and integration into business systems, aided by the make*Sense* process modeling approach and tools. On the other hand, make*Sense* may improve profits by opening up new and creative ways of using WSNs, or optimizing the usage of existing WSN deployments.

Already in the project itself, we looked at three scenarios: *ventilation on demand,* where we save energy by turning off ventilation of rooms when it is not necessary, *solar panel maintenance,* where we monitor the performance of individual solar panels and prevent them from overheating, and *condition-based maintenance for vessel engines,* where we sketched vibration-based health monitoring of engines to schedule predictive maintenance along with the complex spare parts logistics. All of them are real business challenges at Acciona, the partner in make*Sense* playing the role of an industrial end user. The breadth of the scenarios already demonstrates the big potential to use WSNs in many business domains. The make*Sense* approach could act as a vehicle to realize scenarios like this.

As demonstrated in the project, one particular area of interest for WSNs is the energy domain. Sensors and actuators can be used to optimize energy consumption and thus save energy costs. One example of employing WSNs for this purpose was given in the make*Sense* ventilation scenario. In this case, energy costs for heating and/or cooling could be avoided by ventilating meeting or dorm rooms (as showcased in the trial in Cadiz) on demand and only when needed.

[2]Report at idtechex.com
[3]Report at bccresearch.com
[4]Report at culrav.org
[5]Report at researchandmarkets.com

5.2.3 Focus Application Area: Electricity Markets

Among the drivers that contribute to a heightened interest form the energy domain are recent developments such as the focus on smart grids, variable energy pricing, a shift to alternative sources of energy, and decentralized energy production. Further contributing is a traditionally low degree of automation and a low rate of deployment of sensing and actuation technology in today's electricity grids, which leaves room for improvement. Given those levers, we will in the following present ways in which companies can benefit from WSNs.

One of the key characteristics of electricity grids is that the supply must match the demand at any given time. As providing additional energy in times of high consumption can be quite costly for utilities, there is an interest in demand response and load balancing schemes that aim at managing energy demand. These schemes may involve shifting loads, buffering loads, turning of consumption completely or even increasing energy consumption if there is too much energy in the grid. The capability to shift loads can either lie in the hands of the consumer or with the utility. Sometimes a distinction is made between those two types of control, labeling the former demand response and the other load shifting.

The more energy can be shifted on the demand side, the greater is the potential for the utility to reduce costs, industrial energy consumers, such as breweries or manufacturing plants are therefore of prime interest for applying demand response schemes. As an example, heating, air condition or refrigeration may be turned up in times of low energy demand, delaying the need for energy in times peak usage.

Some technologies for implementing demand response are available, while others are being developed. If implemented, they normally rely on control systems at the demand side that turns systems on and off (or sets process parameters that result in lowered or increased consumption) in response to variable energy prices or requests by providers. Usually, preplanned load prioritization schemes exist to steer demand. Conditions for load shifting are specified in contracts, so that the consumer is protected from unreasonable shutting down of devices.

Despite the benefits for energy consumers and providers alike, demand response is not widely used today. One can argue that this is partly due to the unawareness of energy consumers of their marketable load shifting potential. Furthermore, intelligent systems to steer energy consumption in complex setups are lacking. WSNs and an approach such as presented by make*Sense* that gives business experts insights into and programmability of sensors and actuators may facilitate the implementation of demand response schemes.

WSNs can support demand response in a number of ways. Very straightforward schemes can be implemented that simply turn on or off actuators based on pre-negotiated terms, while checking sensor readings to makes sure conditions are met. However, WSNs can also be used to allow for more sophisticated, innovative demand-response scenarios, as described in the following processes:

Identify load shifting potential Identify the available load shifting potential of a participating company. This can be done either a priori, e.g. periodically or when a company signs up for demand response, or the potential can be calculated on the fly to determine the currently available load shifting potential. To determine this value, sensor readings (e.g. the current temperature, current production speed), data about allowed states (e.g. a minimum temperature that must not be violated), and additional data from business systems such as production planning need to be integrated. Using this data, it can be determined when and which amount of energy consumption can be shifted, reduced or also additionally consumed.

Determine costs for load shifting Using data from business systems and maybe even scenario simulations, the participating company can determine the minimum price it needs to receive if it were to shift load so that it can be done economically. Opportunity costs need to be taken into account. For more stable demand response schemes based on pre-negotiated values, the minimum price for offering this flexibility in consumption can be calculated.

Trading of load shifting potential The identified shifting potential can be reported to a broker or trading platform. Once a customer is found, the company can receive the parameters of the deal, e.g. the negotiated price, the timing and volume of the load shift that is requested. The traded shifting potentials can also involve some flexibility, e.g. a deal can be made to reduce consumption by 20kWH for two hours in a four hour period.

Determining a response strategy Once a load shifting potential has been traded or requested by a supplier, the participating company needs to determine the best, usually the cost-optimal strategy to meet the requested shifting goal. Current sensor readings and data from business systems such as production planning can be used to determine a good response strategy. For example, the shifting goal can be reached by turning off the heating and slowing down conveyor belts by 10%. Incorporating current sensor readings in this step allows to base the decision on the actual state of the company and not to miss opportunities to meet the shifting goals at the lowest possible cost.

Execute / control load shifting The strategy devised in the previous step is carried out by setting actuator parameters and update data in the business systems according to the calculated values.

Monitor energy consumption and evaluate load shifting Use actuators and sensor readings to monitor environmental conditions, energy consumption and the state of energy consumers to ensure that the demanded load shifting goal is met and to react to changes in energy consumption. If changes are detected that threaten meeting the goal, back up plans can be identified and executed, to meet the shifting goal in an alternative way.

Aggregate shifting potentials An additional player in demand response schemes can be a dedicated company that contracts and manages load shifting potentials from multiple users and sells the aggregated potentials to utilities. In order to be able to fulfill this task, the load shifting trader needs to be able to identify the current load shifting potentials of all participants, to determine strategies to distribute the requested potentials over various participants, to issue signals when and how much energy consumption to shift, and to monitor whether participants are indeed meeting their shifting goals.

Further possibilities for applying WSNs and process management capabilities exist in the domain of grid management. Currently the portfolio of technologies used for grid control and in control centers is very diverse. A lot of the technology employed is quite old, e.g. in Germany

(60–70 years), and not automated. There is hardly any possibility to flip switches remotely, so technicians typically have to drive to power distribution stations in case of problems or for regular maintenance. There are currently efforts underway that aim to introduce sensing technology more widespread in the grid, demonstrating the pressing need for changes.

New forms of energy production that are often decentralized require even more sophisticated ways of grid control. Smart meters that are beginning to be installed in consumer households serve as additional sensors reporting fine-grained energy consumption. An approach as developed by make*Sense* could help to make the work in the grid control centers easier, by offering process views and integrating different sensors and actuators.

By using WSNs and automating processes, there could be large cost reductions in this domain. However, the deregulated market and complex makeup of stakeholders make rolling out new solutions challenging. As for the demand response scenario, lower market entry barriers exist when interfacing industrial energy consumers with utilities, as a smaller set of stakeholders is involved. An adoption of WSNs and process modeling might therefore start in this area.

5.2.4 Other Application Areas

Some other application areas have been proposed by EIT's technology transfer coaching & advisory program who engaged with the project. Committee members made a few suggestions for possible markets:

- home automation systems, especially towards safety and security (where current systems are not yet wireless based);
- insurance companies and other bodies (such as TÜV in Germany) involved in safety and security: for example protecting art works in museums, ensuring a certain level of safety in the workplace);
- distributed control systems in general (and SCADA in particular);
- more generally software systems for managing plants and warehouses.

The committee noted that some of the above mentioned markets are quite large and already companies operating there are large too but also mentioned that there is no large market without a number of niches associated to it and companies operating within the niches.

5.2.5 Example Demonstrator Use Case: Ventilation on Demand

In this section we give a detailed technical description of the the make*Sense* main demonstration scenario, *ventilation on demand* (as shown in Figure 5.1) and further describe all the business benefits an end-user gets by deploying a make*Sense*-enabled system. As ventilation on demand is only an example use case and the make*Sense* approach can be applied to any WSN system, similar benefits can be expected in other usage scenarios.

The process deals with smart, automatic ventilation of rooms that are use with prior reservation. The idea is to optimize the ventilation by only ventilating when necessary, i.e. when the air quality is bad and the room is used, thus saving energy, as the fed in air does not need to be heated in winter or cooled in summer. The technical process running in a

WSN is integrated with three back-end processes running in external business systems: a room reservation system, a system delivering the current energy price and a system for handling fire alarms. Prices are fed in to save cost by adjusting air quality thresholds. The assumption is that the operator of the rooms is a large organization that does not pay a flat price per kWh, but is subject to fluctuating energy prices[6]. The fire alarm system makes additional use of the CO_2 measurement in the technical process at no additional sensor hardware cost.

After describing the technical details of the process, we will elaborate on the business value that such a make*Sense* deployment brings to the organization operating the rooms.

Technical Description of the Process

The process model shows the intra-WSN process in the middle. This is the part that is compiled by the model compiler, which generates MPL code that runs both on the WSN gateway and on a subset of nodes. Note that the model does not include fixed data on which room is reserved and managed. The model is a blueprint that is instantiated for each meeting. Only at run-time are the appropriate sensors and actuators in the specified rooms selected by dynamic target expressions. The process is supposed to pre-ventilate rooms 15 minutes before the meetings start to ensure optimal air quality.

The top-level process, i.e. the reception of the start message, the initial stopping of the ventilation and the final sending of the *end of meeting* message is deployed on the WSN gateway and runs there. In contrast to that, the two sub processes represent independent logic that runs on WSN nodes directly. They represent two independent, but coupled control loops.

The sub processes depicted above periodically reads the presence sensor(s) in a room. The loop runs locally on the presence sensor WSN node itself. The sensor nodes are selected by both a static target and a dynamic target. The static target represents a selection done at design time. Here it is used to select the correct sensor type (presence sensors). However, the decision in which room the presence sensors should be used, is reflected by a dynamic target, accessing dynamic state information of the individual process instance at run-time. When no presence is detected any more, the process terminates, control is given back to the process on the WSN gateway and this terminates itself as well as the second sub process.

The sub process depicted below runs on the selected room's actuator node. It (remotely) queries the CO_2 sensor(s) and drives the actuator accordingly. The process also periodically receives the current energy price. The decision to turn the ventilation on or off depends on the current air quality (approximated by the CO_2 measurement), whether a meeting is scheduled, the actual presence of people in the room and the current energy price. The effect of the energy price would be to shift the (hysteresis) thresholds on when to turn the actuator on or off [7]. For the sake of readability, we do not depict the complete condition expressions in the diagram. The conditions on the outgoing branches of the BPMN gateway (decision) after the *sense co2* task are actually more complex.

The intra-WSN process is integrated with three external, IT-supported business processes. The first is the reservation system for the management of the meeting rooms. This is the leading

[6] Either because of a special contract with its energy retailer or because it procures its energy directly from an energy exchange.

[7] Ventilating more when energy is cheap and ventilating less when it is expensive, of course within the corridor of comfort for the users of the rooms.

system in our scenario. For each reservation, an instance of the process is created. When this instance sends a room set-up message to the intra-WSN process, containing the correct meeting data (like start and end time and room id), this creates a correlated instance of the intra-WSN process. When the intra-WSN process instance detects that there are no more people in the room, it terminates, but sends an *end of meeting* message back to the corresponding reservation instance, which hence leads to its termination.

Further, the intra-WSN process receives current energy prices from the second external system as described above. Finally, the same WSN set-up is also used to determine irregularly high CO_2 values which are a strong indication of a fire breaking out. In this case, a message is sent to an alarm system (the third integrate external system) that triggers a work flow to handle a fire alarm.

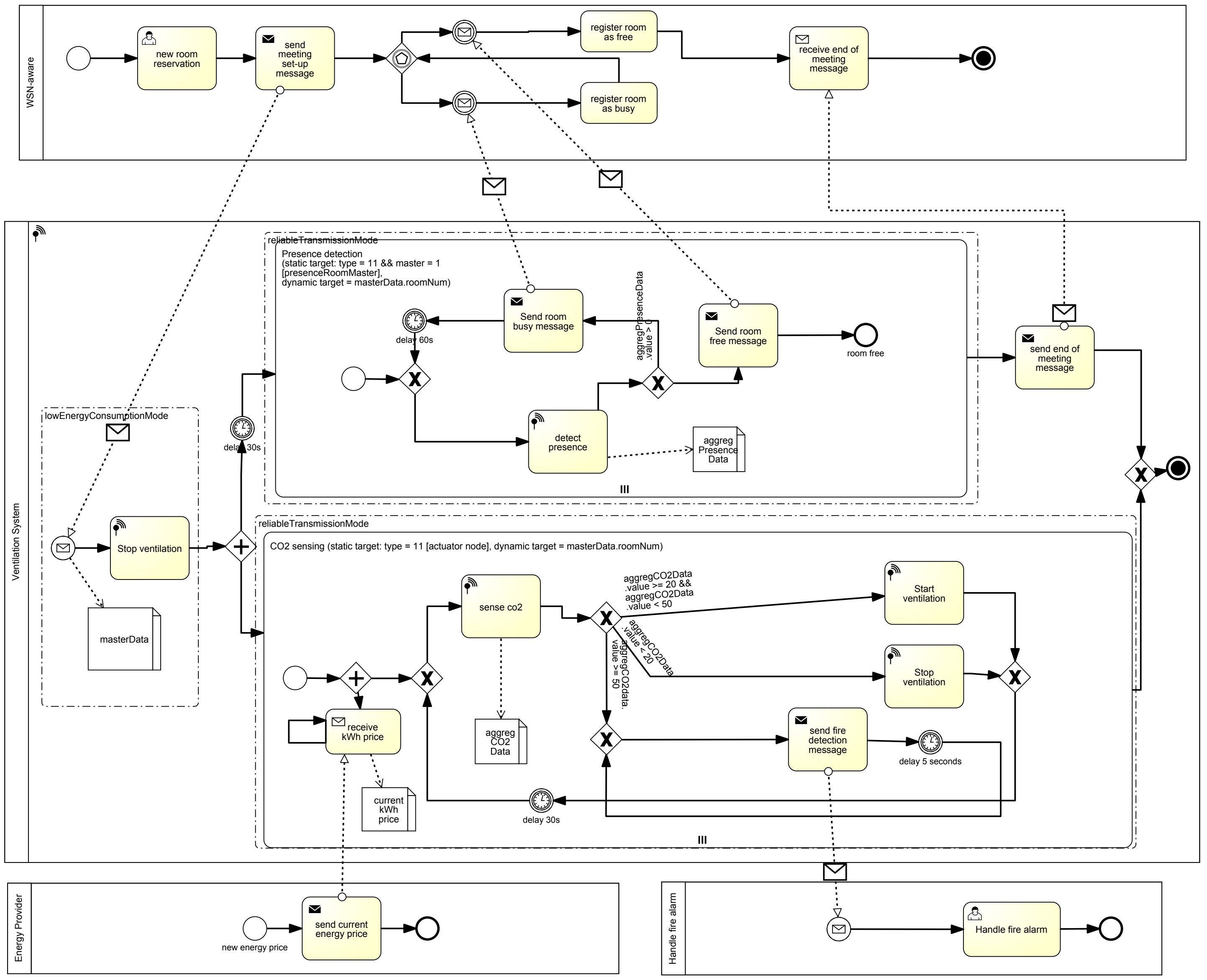

Figure 5.1: "Ventilation on Demand" Process Modeled in Modeling Applicaton V1.0 with two external WSN-aware processes

Business Benefits for End-Users Demonstrated by the Ventilation on Demand Scenario

The final version of the ventilation on demand scenario that the make*Sense* consortium was able to run end-to-end both in simulation and on real hardware in the lab highlights many of the business benefits that end users can achieve by adopting the make*Sense* approach to configure, deploy, and operate wireless sensor networks.

Enablement. If an organization deploys WSN nodes that offer their functionality as concrete make*Sense* abstractions, domain experts can create the WSN logic on their own, provided they have a basic understanding of BPMN and receive a short training or consult brief documentation of the make*Sense* approach to modeling. In our example, a technical person from the organization managing the meeting rooms could define the logic by himself, without needing expensive services form the equipment vendor.

Adaptability. Should the technical process in the intra-WSN process need some adaptation, it can be done by the domain experts themselves. In our example, the system is configured to pre-ventilate the rooms prior to meetings. Should the operators realize that the 15 minute pre-ventilation period is not appropriate (it might be too long and thus waste energy), they could change this parameter by themselves without needing external experts to re-configure or re-program the system. This is both true for small configuration changes or larger re-configuration of the logic.

Extensibility. The make*Sense* system is extensible along multiple dimensions, e.g.

- More sensors of the same kind could be added to the system to extend the reliability of sensor readings (e.g. more than one PIR sensor[8] per room). The sensors will integrate automatically with no further re-modeling needed.
- Different sensors could be added to sense new data (in our example e.g. temperature sensors).
- New logic can be implemented (e.g. to take into account also the temperature or drive the window blinds)
- New interfaces to external systems can be added (just as we added the interfaces to the energy provider and the fire alarm system after the initial modeling).

No vendor lock-in. The make*Sense* system is designed to be open. Hardware can be provided by different vendors. No (expensive) deployment services of a specific vendor need to be purchased. If in our example, the whole ventilation control system would be bought from a single vendor (like Siemens, Honeywell, GE, or the like), it is very likely that the vendor would provide a whole package of hardware and installation and configuration services, causing unnecessary delay and cost (as the domain experts and system engineers have to establish a common understanding of how the system should operate).

Turn-key integration. The make*Sense* tool chain ensures that the intra-WSN process is directly integrated with the WSN-aware processes. Code for sending and receiving message payload (through serialization and de-serialization) is generated automatically for all

[8]Passive Infra-Red presence detection sensor

defined messages that are exchanged between the WSN-aware process and the intra-WSN process. In our example, the four types of messages exchanged between the WSN-aware process of the meeting room reservation and the intra-WSN process are managed automatically. This shifts the problem of integrating an embedded system with an IT system (which is difficult, as the former is very resource constrained while the latter often uses large data structure such as XML documents) to integrating two IT systems, the BPMN execution engine and the other IT system, which share similar design principles. Additionally, such interfaces often change, which can now be handled internally by the operating organization, again not requiring costly external service. In our example, e.g. software systems handling catering, maintenance or cleaning of the rooms could be easily integrated.

Self-adaptation during run-time. As the system can be tuned already in the model to self-adapt to the current execution state at run-time, benefits include

- Higher energy efficiency of individual nodes, leading to longer battery life time and thus lower cost of operation.
- As the system puts itself dynamically to low latency mode when the current execution state requires this, processes will become more reactive (e.g. the user interfaces will have a low latency).
- The same applies for reliable transmission mode, which ensures that no messages are lost and the processes run reliably.

Precise, intuitive business insight. The business-level monitoring of the process execution allows for transparency of the execution state. An operator can look into any running process instance running and assess the current execution status by reviewing the active tokens in the process. In the example, the operator could see whether the room is currently occupied and if the ventilation is currently turned on or off.

General benefits of WSN deployments. On top of the make*Sense*-specific benefits for the business, the general benefits of deploying WSNs come for free:

- Better coupling of the physical world (of atoms) and the virtual model of an organization as it exists in the IT system (world of bits). In our example, the reservation systems knows in real-time if a meeting room is really occupied or not (not relying solely on data in the system).
- More timely reaction of business processes to real-world events. In the example, no one has to confirm the meeting room status as it is assessed automatically.
- Higher degree of automation of processes through elimination of manual data entry. In the meeting room example, the ventilation does not have to be regulated manually.

5.3 Impact on Society

make*Sense* has the potential to dive WSN adoption, therefore amplifying the effects that WSNs generally have on society.

WSNs have the possibility to contribute to the socio-economic goals outlined in i2010 that include ICT solutions for monitoring health and well-being, technologies for environmental sustainability and energy-efficiency as well as applications for better inclusion and independent living of all citizens. The make*Sense* application domain i.e. energy management directly addresses energy-efficiency contributing to environmental sustainability. Many of the innovative WSN applications described in the CONET and Embedded Wisents Roadmaps including intelligent pills, intelligent waste management systems, smart grids, congestion-free road traffic, self-learning children watching sensor networks as well as sensor networks for human augmentation and enhanced human-animal interaction may help to fulfill these ambitious goals in i2010.

Many of these applications would enhance citizen's quality of live but do not necessarily directly lead to monetary profits. In order to nevertheless realize these applications, it is of major importance that they are easy, i.e. cost-effective and in short time, to develop. make*Sense* will thus help to realize many applications beneficial for society and individuals that would not be realized if their development was cumbersome and hence time-consuming and expensive. If the results of make*Sense* are actively applied, then we will be able to have an easier programming of sensor networks, which will lead to their widespread usage even by simple users. This will have a snowball effect, which will enable their adoption in different dimensions e.g. in social applications, eHealth, household appliances etc. As such SMEs will be able to create new applications with a low learning curve on programming, and this could lead to creation of new jobs.

5.4 Conclusion

To summarize, while the application of WSNs and process modeling is by no means limited to the energy domain, scenarios like demand/response, grid management or simple energy saving processes already demonstrate that this approach is beneficial and drastic cost savings can be obtained. For the more complex scenarios, WSNs alone might not be able to achieve the same benefits as an integrated approach and tool set as offered by make*Sense*, allowing business experts to fully reap the potential of sensing and actuating technology. Therefore, SAP has patented its part of the make*Sense* approach, i.e. application modeling and compilation, and concepts from make*Sense* have been incorporated in the *SAP HANA Platform, IoT Edition*[9], a new product targeting at integrating devices from the Internet of Things with SAP systems.

[9] http://www.sap.com/iot

6 Contact details

>makeSense
Easy Programming of Integrated Wireless Sensor Networks

Figure 6.1: The project logo

Contract Number: 258351
Full name: Easy Programmming of Integrated Wireless Sensor Networks
Project web site: `http://www.project-makesense.eu`
Project coordinator:
Name: Thiemo Voigt
Institution: SICS Swedish ICT AB
e-mail: thiemo@sics.se

Partners:
Universität zu Lübeck (UZL)
Contact: Kay Römer, romer.kay@googlemail.com
Universitá degli Studi di Trento (UNITN)
Contact: Gian Pietro Picco, gianpietro.picco@unitn.it
SAP AG
Contact: Patrik Spieß, patrik.spiess@sap.com
Acciona Infraestructures S.A.
Contact: Patricio Moreno Montero, patricio.moreno.montero@acciona.com

Videos:
Within the project, a new approach is taken to ease the programming of Wireless Sensor Networks by combining the world of business processes and the world of sensor networks. This is shown in video that has been produced within the project. See `make`*Sense* marketing video at YouTube `http://www.youtube.com/watch?v=n4ospuvPrWA`
From SICS Openhouse 2012 `http://youtu.be/xeXhmskw_00`

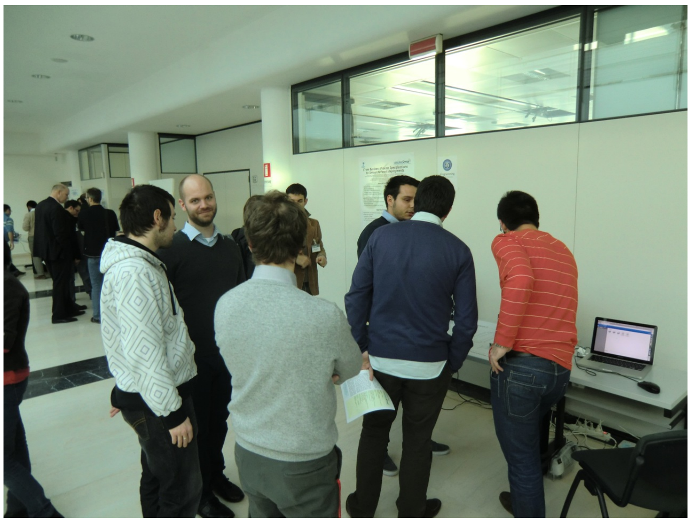

Figure 6.2: Presentation of make*Sense* at EWSN 2012

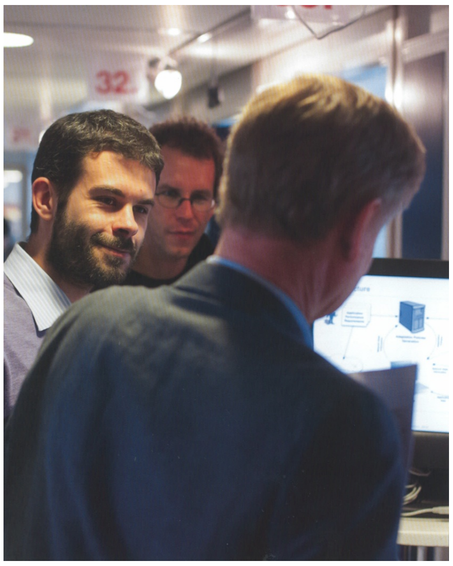

Figure 6.3: Presentation of make*Sense* at SICS Openhouse 2012

Bibliography

[1] L. Mottola and G. P. Picco, "Programming wireless sensor networks: Fundamental concepts and state-of-the-art," *ACM Computing Surveys*, vol. 43, no. 3, Apr. 2010.

[2] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 37–54. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.14

[3] F. Daniel, J. Eriksson, N. Finne, H. Fuchs, A. Gaglione, S. Karnouskos, P. M. Montero, L. Mottola, F. J. Oppermann, G. P. Picco, K. Römer, P. Spieß, S. Tranquillini, and T. Voigt, "makesense: Real-world business processes through wireless sensor networks," in *4th International Workshop on Networks of Cooperating Objects for Smart Cities 2013 (CONET/UBICITEC 2013), held in conjunction with CPSWeek*, Philadelphia, USA, Apr. 2013.

[4] C. Decker, T. Riedel, M. Beigl, L. de Souza, P. Spiess, J. Muller, and S. Haller, "Collaborative business items," in *3rd IET Int. Conf. on Intelligent Environments*, 2007.

[5] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of Web services," *IEEE Trans. on Service Computing*, vol. 3, no. 3, 2010.

[6] L. Mottola and G. Picco, "Logical Neighborhoods: A Programming Abstraction for Wireless Sensor Networks," in *Proc. of the Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, 2006.

[7] N. Finne, J. Eriksson, N. Tsiftes, A. Dunkels, and T. Voigt, "Improving sensornet performance by separating system configuration from system logic," in *Proceedings of the Sixth European Conference on Wireless Sensor Networks (EWSN2010)*, Coimbra, Portugal, 2010.

[8] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco, "Programming Wireless Sensor Networks with the TeenyLIME Middleware," in *Proc. of the* 8^{th} *ACM/USENIX Int. Middleware Conf.*, 2007.

[9] M. Ceriotti, L. Mottola, G. P. Picco, A. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon, "Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment," in *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 277–288.

[10] R. Sutton and A. Barto, *Reinforcement learning: An introduction.* The MIT press, 1998.

[11] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "pTunes: runtime parameter adaptation for low-power MAC protocols," in *ACM/IEEE Int. Conference on Information Processing in Sensor Networks (IPSN)*, 2012.

[12] M. Research, "The connected life: A usd4.5 trillion global impact in 2020," Machina Research, Tech. Rep., 2012.

[13] S. Karnouskos, P. Marrón, G. Fortino, L. Mottola, and J. Martínez-de Dios, "Markets for cooperating objects," in *Applications and Markets for Cooperating Objects*, ser. SpringerBriefs in Electrical and Computer Engineering. Springer Berlin Heidelberg, 2014, pp. 99–115. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-45401-1_5