A WSAN Solution for Irrigation Control from a Model Driven Perspective

Fernando Losilla, Pedro Sánchez, Cristina Vicente-Chicote, Bárbara Álvarez, Andrés Iborra

fernando.losilla@upct.es
Universidad Politécnica de Cartagena (Spain)

Abstract. Wireless Sensor and Actor Networks (WSAN) constitute a growing research field in several engineering areas. One very interesting domain of WSAN application is precision agriculture, and in particular, the automatic control of tree irrigation depending on sap flow levels. Nowadays, the software development process followed in these kinds of applications is largely dependent on the platform where the final implementation is done. Consequently, commonly desired attributes such as flexibility, reuse and evolution are relegated to a second level of priority. Nevertheless, the growing interest in WSAN has been led to advances from different points of view: new application domains, new middleware, new simulation environments, and so on. In spite of all these advances, WSAN development is today needed of concrete mechanisms that make easy the software generation process. This paper summarizes our contribution in this field from two points of view: as an agronomic solution and as new opportunities for affording the construction of these systems taking into consideration the most recent advances in software engineering.[1].

Keywords: Model driven approach, Domain Specific Languages, Precision Agriculture.

1. Introduction

Recent technological advances have led to the emergence of wireless sensor and actor networks (WSAN). These networks are able to observe the physical world, process data, make decisions on the basis of these observations, and carry out concrete operations on the environment.

WSAN applications constitute a new way of acquiring data and controlling the physical world, and therefore they are very useful for developing a broad range of

[1] This work was partially supported by the Spanish CICYT project MEDWSA, reference TIC2006-15175-C05-02 and the Regional Government of Murcia Seneca Program, reference 02998-PI-05.

applications such as [1]: environmental monitoring, health control by tele-medicine, precision agriculture, military operations, transport tracking, etc.

Besides, model driven software development (also known as model driven engineering or MDE [2][3]) solves the common problem of the strong dependence of the software process on the final execution platforms. The MDE approach focuses on the use of models through which software applications can be described while taking independent platform concepts into account and promoting the definition of semi-automated transformation mechanisms from those models in final code.

These approaches to the development of WSAN based applications entail the following research activities: (1) to adopt and apply a method that guides the domain engineering process from which the concrete product line study will be derived; (2) to define a domain-specific language with which models can be produced by representation of concepts from that domain; (3) to select an execution infrastructure-independent language with which all architectural design decisions for the product family can be represented; (4) to define the transformations between models taking into account the meta-models (that is, models that define a language for expressing other models) identified for each language utilized in the process; and (5) to develop tools which provides support for the whole process.

In this paper we describe with detail the adopted solution for a well know problem of precision agriculture: control irrigation by measuring the tree sap flow. This example has allowed us to put in practice the languages, tools and developed software artefacts.

For a good understanding of the work, we first describe the case study in section 2. Section 3 gives an introduction to the concepts that could enhance the development of WSAN applications: the model driven software development perspective. Sections 4 and 5 present the case study solution from a model management point of view. Section 6 gives the related work and section 7 conclusions and future works.

2. Case study description

The MITRA test bed consists of thirty TinyOS-based nodes (see Figure 1) deployed in an almond orchard located in the region of Murcia (southeast of Spain), where the climate is semiarid Mediterranean. Given the shortage of water in this region, the prime objective of the system is to regulate tree irrigation according to water stress, that is, to water the trees only when required. Water stress is measured using the heat pulse compensation method [4] which consists in generating a pulse of heat throughout an axial line of the tree trunk using a resistor element. Then the temperature evolution is measured above and below this line to determine the sap flow and hence the water stress.

Figure 2 shows the hardware architecture of these MITRA nodes. The microcontroller, together with a driver, generates an electrical pulse and applies it on the heater. The heater is a very thin resistor made up of a nicron wire inside a 2 mm steel tube. To calculate the sap flow, it is necessary to measure the temperature at different depths of the tree trunk. To get this information, MITRA nodes provide two temperature sensors. In addition, MITRA nodes are equipped with a port for connecting up to eight additional sensors (e.g. luminosity, temperature, humidity of the soil, and air pressure).

The microcontroller processes the signals provided by the temperature sensors in order to calculate the sap flow. This datum is reported via WiFi every three hours (for energy-efficiency considerations) to a simple PC, which controls the irrigation process on the basis of the information received from all the deployed MITRA nodes.

When the PC detects that some water is needed, it sends a “start irrigation” order to the watering system via WiFi, and conversely, when it detects that the trees are sufficiently watered, it sends a “stop irrigation” order. The section 4 explains how the MITRA system was implemented using the tools and the methodology presented in this paper.

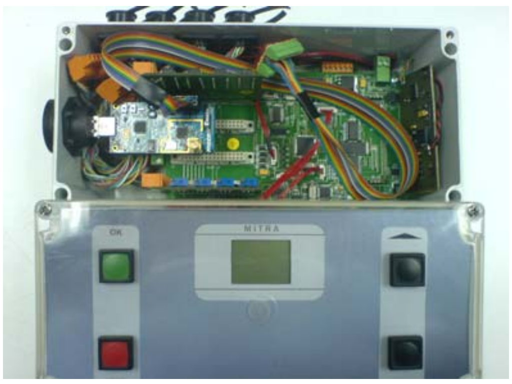

Figure 1: The MITRA node.

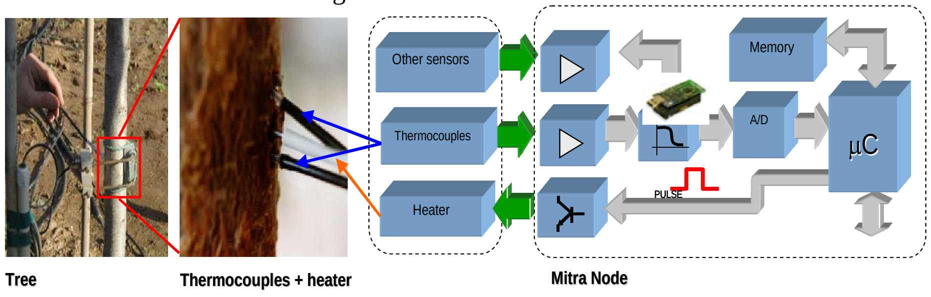

Figure 2: MITRA node hardware architecture.

3. A model driven perspective for developing WSAN applications

Model Driven Engineering (MDE) promotes software development centred on the concept of a systematic use of a *model*. In this approach, models play the most important role because they are the chief artefact guiding not only the development and documentation of the software but also its management and evolution. In MDE [2] models are created on the basis of formal *meta-models* which describe complementary views of a system at different levels of abstraction.

In MDE, software can be described by means of models at high levels of abstraction, and then these models can be compiled into other representations closer to the implementation level (the executable code) by applying predefined transformations. Because formal descriptions of the meta-models are used, both the building of models and the transformations between them can be done using tools that raise the level of software process automation.

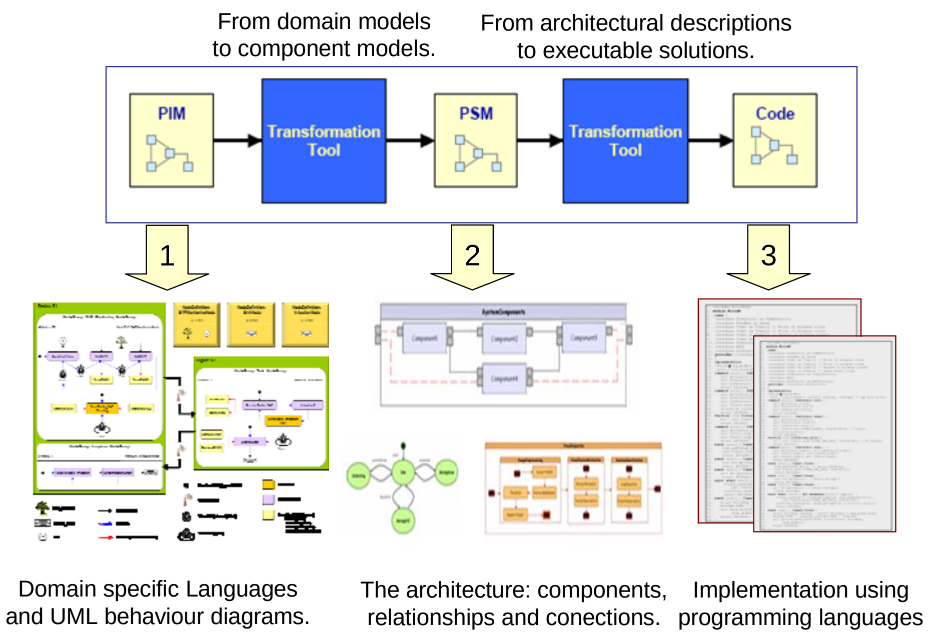

Figure 3: Model Driven Engineering using MDA abstraction levels.

The MDA (Model Driven Architecture) proposal [5] from the OMG (Object Management Group) fits the MDE perspective. MDA proposes three abstraction levels for models of a system (see Figure 3). At the highest level, the platform-independent model (PIM) gives a platform-independent view of the system. This level can be described using for example domain specific languages. The successive lower levels capture platform-dependent features (PSMs).

These PSMs may or may not correspond directly to the code level. MDA promotes the description of software by means of PIM models unrelated to any concrete technology. In a subsequent step, the user selects the final implementation platform to which the upper models will be converted. MDA proposes a combination of the above approach with standards such as MOF (MetaObject Facility), UML (Unified Modelling Language) and XML (Extensible Markup Language) in order to achieve interoperability among the tools involved in the software development process. The MOF specification defines a language and a set of standard interfaces that can be used to define and manipulate a set of meta-models and their corresponding models.

There are several commercial tools for creating, accessing and modifying these meta-models through MOF interfaces. The most representative is the Eclipse Modelling Framework, which has been the selected tool for supporting our work.

A Domain Specific Language (DSL) [6] is a language that offers very good expressive power to capture the requirements of a particular domain at a PIM level. There are many advantages to considering languages of this kind. A DSL provides concrete abstractions to represent concepts from the application domain and offers a natural syntax notation and mechanisms for optimization and error checking.

One of the key factors when defining a DSL is its ability to provide users with a set of modelling primitives very close to the domain abstractions. The DSL we have defined provides concrete and precise constructs for defining WSAN applications on an independent implementation platform such as MDE promotes.

This WSAN-DSL by the authors (an example in next section) provides constructs for specifying both structure and dynamics of WSAN applications at two different levels: node level and region level. It considers a WSAN as an aggregation of regions where a region is seen as a set of atomic nodes and sub-regions with similar characteristics.

The use of regions allows for the building of heterogeneous WSANs where not all nodes have to implement the same functional subsystems (i.e. localization, routing and other specialized processing), but they can be grouped into regions according to what subsystems they implement and how they do it.

Figure 3 summarizes the approach adopted. The picture depicts the separation between PIM and PSM at different abstraction levels. Models represented at a certain level are then transformed into models at lower abstraction levels by means of tools which supply the process with automatic support.

Each level contains a representation of the system using a meta-model that provides formal support for a well-defined modelling or implementation language (model management perspective). Next section presents the obtained models for the case study taking into account the referred software artefacts.

4. General description of the solution

The first step in the proposed methodology for WSAN application development is to build a model of the target system using the WSAN DSL. This high level of abstraction modelling language (meta-model) provides all the concepts and relationships commonly used for specifying WSAN applications.

As noted earlier, a new graphical modelling editor which allows WSAN domain experts to graphically describe the structure and the behaviour of their systems has been developed on the basis of this WSAN meta-model. The MITRA system model depicted using this DSL is shown in Figure 4. Two different regions have been defined in this model. The first region includes two node groups, one representing the MITRA nodes deployed in the almond orchard (SAP Monitoring NodeGroup) and another representing the irrigation control node (NodeGroup containing only one node). The second region contains only one NodeGroup with a single node (Sink node) with the responsibility of transferring data between the WSAN and a computer.

The behaviour of the three different types of nodes (node groups) defined in the model is specified following a three-step process: (1) select the sensors to be read, (2) select the node activities (from those provided by the WSAN DSL), and (3) link all these elements together according to the rules specified in the meta-model.

As can readily be appreciated from the figure, the behaviour of the SAP Monitoring NodeGroup contains two entirely uncoupled activity sets, one describing the sensing loop and another describing how the collected data are sent to the Sink NodeGroup via WiFi. The model transformation which translates WSAN DSL models into generic component models uses this information to place uncoupled activity sets (detected in the WSAN DSL model) into orthogonal state regions in the transformed generic component model (Figure 3, step 2).

Finally, the messages required to model component communication are also inferred. For instance, the MITRA system requires two different messages, one containing the sap values sent from the SAP Monitoring NodeGroup to the sink node and another containing the irrigation orders the PC sends to the Irrigation Control NodeGroup.

Once the generic component model has been obtained from the initial specification, a new model-to-model transformation is performed to obtain an equivalent NesC component model. This transformation identifies certain component patterns in the original model and creates the corresponding NesC components, configurations (complex components), and interfaces. The resulting Nesc model is then automatically transformed into NesC code, using the MOFScript model-to-text transformation implemented to support the last step of the proposed methodology.

The final application that emerges at the end of this process has been tested under real conditions (in an almond orchard belonging to the School of Agricultural Engineering of the Technical University of Cartagena, Spain), with successful results. Obviously, the final application code is still far from being optimized since the tools, particularly those relating to model transformations, still need improving. However, this case study has proven the viability of the approach, demonstrating a significant reduction in the effort required for development.

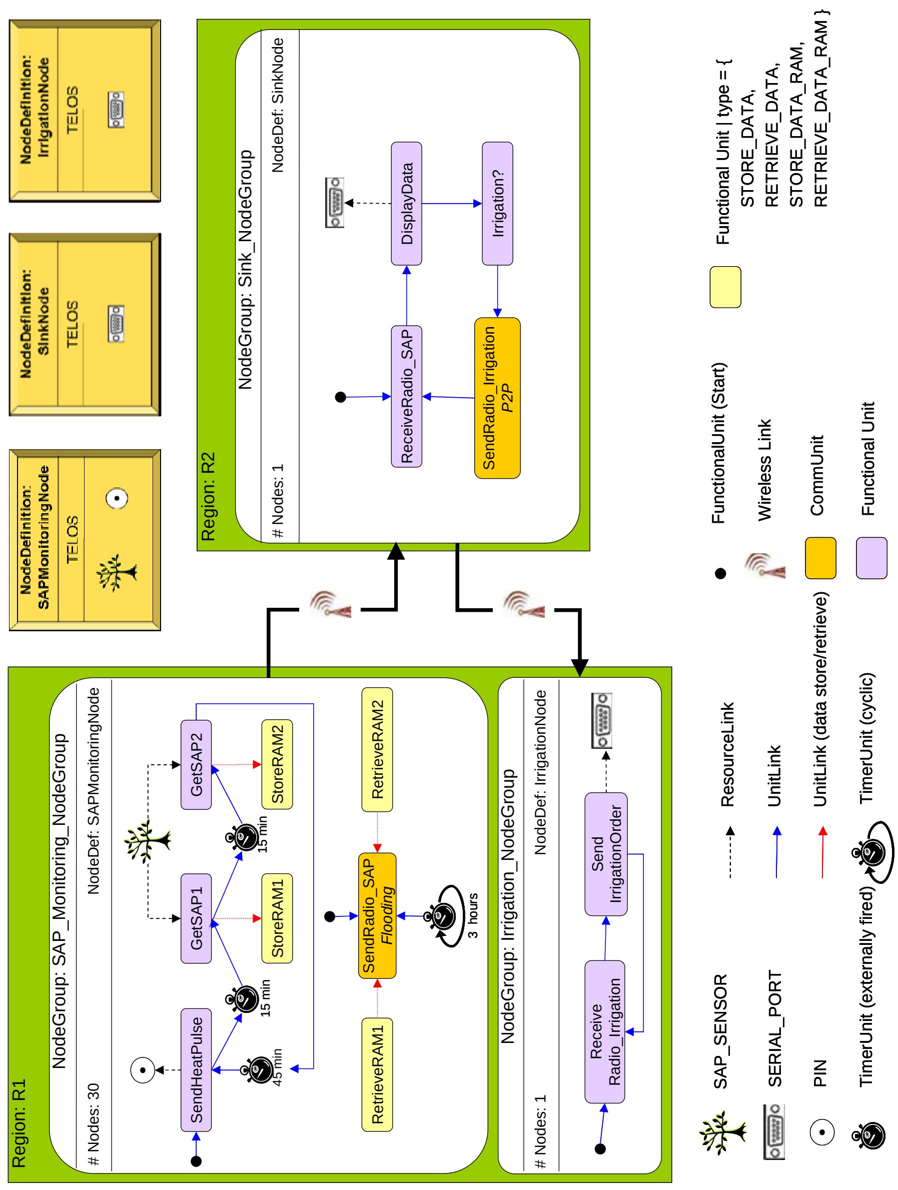

Figure 4: MITRA System model depicted using the WSAN DSL graphical modelling editor.

5. More detail about the transformation process

Figure 4 shows a DSL-description of the functionality of the case study. This representation needs to be translated into a more detailed one that takes into account the services provided and required by each device included in the description. An analogous description of part of the system is given in Figure 5, where an UML Activity Diagram has been used.

Following the UML notation, sent and waited events, and clocks are explicitly represented. It has been considered a unique super-state for the motes which includes two concurrent activity descriptions (left side for sap flow monitoring and right side for periodical sending of data with radio).

It is possible to deduce the component structure described in Figure 6 by considering the set of components and their interfaces. Both Figure 5 and Figure 6 are the input for the compilation process to NesC code. Figure 7 gives a simplified instance of the final code. Some rules need to be considered when generating this code could be the followings:

1. Each referred clock in Figure 5 implies a basic mechanism of starting (for instance, code #29) and attending the associated timeout event (code #53).
2. The sequence described in the activity diagram is followed in the implementation. For example, Sensor.GetSAP1() is done after receiving the end of Timer2. In consequence, the command must be called within the handling of the event (code #43).
3. It is necessary to distinguish different handlers for the event Sensor_Receive_Data with a local variable (code #49...#51).
4. Commands init() and start() must include the initialization of components.

Although we are aware of the difficulty of generating the full code needed to compile the application, however a great percent of code can be deduced both from activity diagrams and components descriptions. Good results can be obtained if appropriate rules are considered in the tool implementation and there exists well documented repositories of components. In this sense, we follow working on the mapping between the different levels of abstraction of the MDE perspective.

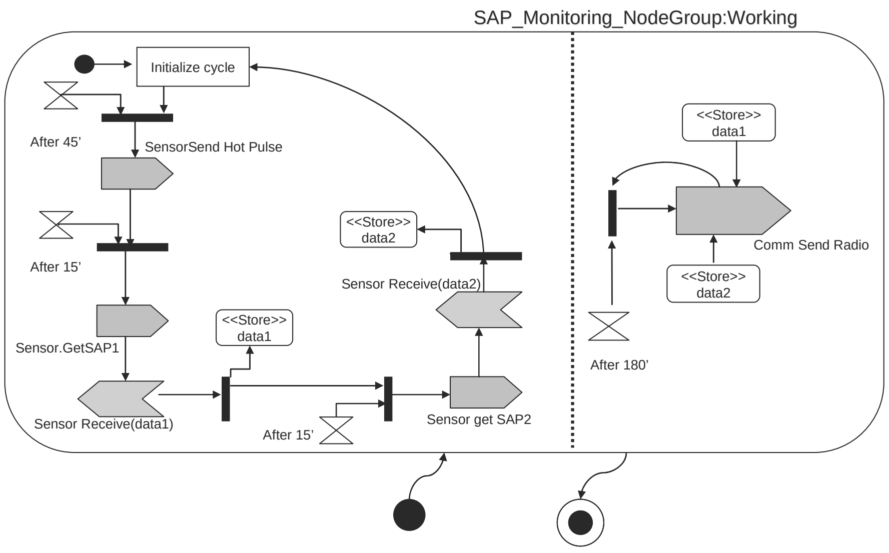

Figure 5: The MITRA System model represented as UML Activity Diagrams inside the state 'working' of the node SAP_Monitoring_NodeGroup.

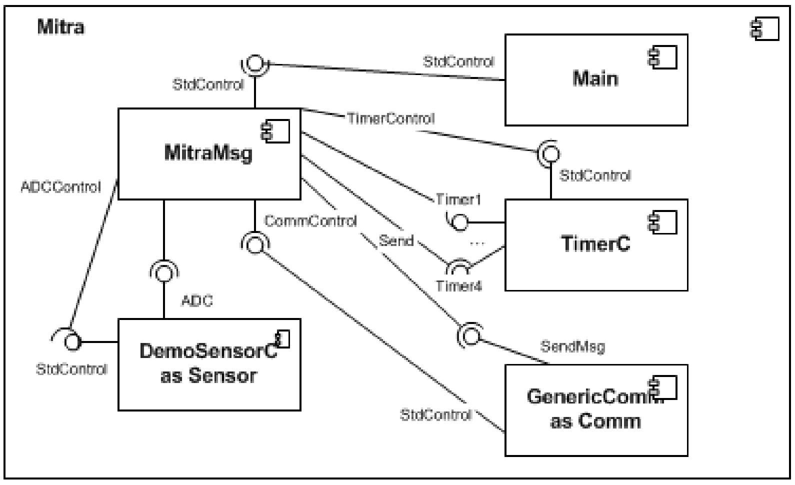

Figure 6. NesC Components and their interconnections.

```
includes MitraMsg;
module MitraM
{uses {
 interface StdControl as CommControl;
 interface SendMsg as Send;
 interface Timer as Timer1; // First 45 minutes clock
 interface Timer as Timer2; // First 15 minutes clock
 interface Timer as Timer3; // Second 15 minutes clock3
 interface Timer as Timer4; // three hours clock
 interface StdControl as TimerControl;
interface ADC;
interface StdControl as ADCControl;}
provides {interface StdControl;}
}
implementation
{TOS_Msg msg_global;
  uint8_t num_reading=1; uint16_t reading1, reading2; // sap flow values
command result_t StdControl.init() {
   call ADCControl.init();
   call TimerControl.init();
   call CommControl.init();
   return SUCCESS; }
void initialize_cycle(){
call Timer1.start(TIMER_ONE_SHOT, 1024*60*45); // 45 minutes}
command result_t StdControl.start() {
   call ADCControl.start();
   call TimerControl.start();
   call CommControl.start();
   call Timer4.start(TIMER_REPEAT, 1024*60*60*3); //3 hours
   initialize_cycle();
   return SUCCESS; }

command result_t StdControl.stop() {
   call ADCControl.stop(); call CommControl.stop();
   call Timer1.stop();  call Timer2.stop();
   call Timer3.stop(); call Timer4.stop();
   return SUCCESS;}
event result_t Timer1.fired() {
    SENSOR_SEND_PULSE();
    call Timer2.start(TIMER_ONE_SHOT, 1024*60*15);
    return SUCCESS;}
event result_t Timer2.fired() {
    call ADC.getData();  // Sensor.Getsap1()
    return SUCCESS;}
event result_t Timer3.fired() {
   call ADC.getData();  // Sensor.Getsap2()
    return SUCCESS;}
async event result_t ADC.dataReady(uint16_t sapFlow) {
     if(num_reading==1) {reading1=sapFlow; (num_reading++%2)+1;
          call Timer3.start(TIMER_ONE_SHOT, 1024*60*15);}
     elsif(num_reading==2) {reading2=sapFlow; initialize_cycle();}
     return SUCCESS;}
event result_t Timer4.fired(){
   struct MitraMsg *message = (struct MitraMsg *) msg_global.data;
   message->RAM1 = reading1; message->RAM2 = reading2;
   call Send.send(TOS_BCAST_ADDR, sizeof(struct MitraMsg),
          &msg_global));
   return SUCCESS;}
event result_t Send.sendDone(TOS_MsgPtr msg, result_t success){}
}
```

Figure 7. Code for the MITRA Application

6. Related work

The foregoing sections showed the benefits of applying a model-driven approach to WSAN application development. The results have been demonstrated by considering a precision agriculture case study. Similar experiments have demonstrated its viability for other reactive systems. But few work-patterns related to this topic have been found in the WSAN domain and none of them use MOF as the underlying substrate. GRATIS II [7], built with the Generic Modelling Environment (GME) meta-modelling tool, constitutes a very well known approximation to the problem, but it does not cover the whole development process. It uses a NesC meta-model and can be used to generate TinyOS configuration components from graphical models and *vice versa*. ATaG [8], also developed with GME, offers a data driven approach for end-to-end application development. In order to model an application, a set of abstract tasks representing types of information processing functions in the system is used along with a set of abstract data items that represent types of information exchanged between abstract tasks. The programmer must supply the code for the implementation of each abstract task and each abstract item of data, but no mechanism is supplied to model this behaviour. Another environment which uses meta-modelling techniques is CADENA [9]. With CADENA, software can be generated not only for WSAN but for any type of application with a component-based architecture, but this generality makes the WSAN software development process excessively complex since domain particular concepts are not used in the functionality description.
The work presented in this paper addresses the creation of a new environment to support the whole WSAN software development process. The resulting environment aims to greatly simplify the construction of applications, making it possible to describe network node behaviour with a small set of concepts relating to the WSAN domain while hiding implementation details from the developer. The application description is currently transformed to TinyOS components in NesC language, but a transformation to other node-level programming environments may be possible provided that they offer a textual or MOF-compliant language for application specification and that they offer an event-driven or thread-based programming model, scheduling schemes, fault tolerance, etc.

7. Summary and future work

This paper has introduced a new model driven method for the development of WSAN applications that allows for the use of a DSL, meta-models and transformations between models. The description of our results has been supported by a WSAN case study. On the one hand MDE offers automation of the development process and platform independence, while on the other hand the study of the WSAN domain approach offers the best possible reuse in software development. Families of software products are organized around an architecture that utilizes the common features of their members and hence offers a set of reusable artefacts. Moreover, a model-driven software development process focuses on defining models independently from

platforms; it also focuses to a greater or lesser extent on the automation of model compilation, and finally on code generation.
Most domain engineering proposals place a great deal of stress on modelling activities as a means of developing product families. The key idea of domain specific languages is to be able to represent the concepts of the application domain directly. In this sense domain specific languages have several advantages over other kinds of general-purpose languages, basically because [6] the constructs are closely related to the concepts of the domain, and these languages usually provide a graphic notation and specialized tools (editors, optimizers, etc.) which gather a great deal of information from the domain. They also provide the opportunity to automate much of the process.

This makes for considerable improvement in the development of reactive systems, particularly in the domain of WSAN applications. We follow working on the implementation of the mentioned tools, the clear definition of transformation rules between models, the demonstration of MDE benefits (such as the independence from the final execution infrastructure) and, finally, the definition of metrics to know the WSAN application time reduction.

References

1. K. Römer, F. Mattern, "The design space of wireless sensor networks", IEEE Wireless Communications, pp. 54-61, Dec 2004.
2. S. Kent, “Model Driven Engineering”, in Proc. of Integrated Formal Methods: Third International Conference, IFM 2002, Lecture Notes in Computer Science, vol. 2335, Springer-Verlag, 2002.
3. S. Deelstra, et al., "Model Driven Architecture as Approach to Manage Variability in Software Product Families", in Proc. of the Workshop on Model Driven Architecture: Foundations and Applications (MDAFA 2003), pp. 109-114, University of Twente, June 2003.
4 P. Becker, “Limitations of a compensation heat pulse velocity system at low sap flow: implications for measurements at night and in shaded trees”, Tree-physiol. Victoria [B.C.] Canada. Heron Pub., Mar 1998 v.18(3) p.177-184.
5. OMG Model Driven Architecture (MDA) Guide v1.0. Available at: http://www.omg.org/docs/omg/03-06-01.pdf
6 J. Estublier, G. Vega, A. D. Ionita, "Composing Domain-Specific Languages for Wide-Scope Software Engineering Applications", in Proc. of MoDELS 2005, LNCS 3713, Springer-Verlag, 2005, pp. 69-83.
7. P. Volgyesi, et al., “Software Composition and Verification for Sensor Networks”, Science of Computer Programming (Elsevier), 56 (1-2), pp. 191-210, April 2005
8. A. Bakshi, et al., “The Abstract Task Graph: A Methodology for Architecture-Independent Programming of Networked Sensor Systems”, in Proc. Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services (EESR'05), Washington, USA, June 2005.
9. G. Trombetti, et al. “An Integrated Model-driven Development Environment for Composing and Validating Distributed Real-time and Embedded Systems”, in Model-Driven Software Development, S Beydeda, M. Book and V. Gruhn (Eds), Springer-Verlag, 2005