Wireless Sensor Network Application Development: An Architecture-Centric MDE Approach*

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

División de Sistemas e Ingeniería Electrónica (DSIE)
Universidad Politécnica de Cartagena, 30202 Cartagena, Spain
{Fernando.Losilla,Cristina.Vicente,balvarez,Andres.Iborra,
Pedro.Sanchez}@upct.es

Abstract. Nowadays, Wireless Sensor Networks (WSN) are a very promising research field since they find application in many different areas. Current proposals for WSN system development are mainly focused on implementation issues and they rarely rely on a Software Engineering methodology which supports their entire development life-cycle. The Model-Driven Engineering (MDE) approach can contribute to solve this problem by allowing designers to model their systems at different abstraction levels, providing them with automatic model transformations to incrementally refine abstract models into more concrete ones. In this vein, this paper presents a MDE approach to WSN application development. Three levels of abstraction have been defined which allow designers to build: (1) domain-specific models, (2) component-based architecture descriptions, and (3) platform-specific models. Automatic model transformations between these three abstraction levels have been designed and, in order to demonstrate the viability of the proposal, a real WSN application has been developed using the implemented tools.

Keywords: Model-driven engineering, component-based software architecture, domain specific languages, wireless sensor networks, Eclipse platform.

1 Introduction

Recent technological advances have led to the emergence of Wireless Sensor Networks (WSN). These systems are able to observe the physical world and to obtain useful information from it. They can process the retrieved data, make decisions on it, and carry out concrete operations on the environment [1]. Nowadays, Wireless Sensor Networks find application in many different domains, such as: environmental monitoring, tele-medicine, or precision agriculture, among others [2]. In 2003 the MIT's Technology Review [3] published a study where WSN applications were cited as *"one of the top ten technologies that will change the world"*. However, current

techniques for implementing this kind of systems seem to be not powerful enough to deal with their growing complexity.

Current proposals for WSN application development are mainly focused on implementation issues. Actually, most of these systems are built from scratch following an experience-based method, which advocates for selecting the most appropriate target platform first, and then the WSN domain-specific operating system (e.g. TinyOS [4]) and programming language (e.g. NesC [5]). The lack of a Software Engineering methodology which supports the entire development life-cycle of these applications, commonly results in highly platform-dependent designs, difficult to maintain, scale and reuse.

The Model-Driven Engineering (MDE) approach can help reducing this dependence of the software development process on the final execution platforms [6]. MDE revolves around models (defined at different levels of abstraction), and automatic model transformations, aimed at incrementally refining models into final application code. Models are defined in terms of formal meta-models (or modelling languages). These include the concepts needed to describe a system (or a set of systems) at a certain level of abstraction, and the relationships existing between them. In order to describe the model transformations, that is, how abstract models are refined into more concrete ones, a mapping between their corresponding meta-models must be defined. Thus, applying a MDE approach requires defining both, the appropriate meta-models and the corresponding model transformations.

This paper presents a MDE approach to WSN application development aimed at improving the flexibility and reusability of their designs. Three meta-models have been defined at different levels of abstraction together with the corresponding model transformations. Designers model their systems using only the WSN domain concepts included in the highest level meta-model. These initial models are then successively refined through model transformations until the final application code is automatically generated.

Before entering into details, the following section presents a motivation example based on a real WSN application for *precision agriculture*, which highlights the lack of flexibility and reusability of current WSN application designs. This is followed by an outline of the research goals and process. Then, the rest of the paper is organized as follows. First, Section 2 briefly presents the platform selected to implement the tools developed as part of this work. Then, the different meta-models and model transformations implemented as part of the proposal are presented in sections 3 to 6. Section 7 reviews some related works and, finally, Section 8 presents the conclusions and some future research lines.

1.1 A Motivation Example

The MITRA WSN application consists of thirty TinyOS-based nodes deployed in an almond orchard located in the semiarid region of Murcia, in the southeast of Spain. Given the of shortage 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 it is needed. Water stress is measured using the heat pulse compensation method. This method consists in generating a heat pulse through the axial line of the tree trunk and

measuring the sap temperature at two different points along this line. Similar temperatures indicate a fast sap flow and this suggests that some watering is needed.

The MITRA application was initially developed using a traditional approach. A new small electronic sap flow sensor was designed and the software to control both, local data processing and wireless communications was implemented in NesC, a component-based programming language for TinyOS-based WSN applications.

The resulting system was highly dependent on the TinyOS-NesC platform and on the custom sap sensor. The lack of flexibility of the design required several changes, both in hardware and in software, to cope with every small change in the application. This led us to search for a more flexible design solution, as described in the following subsection.

1.2 Research Goals and Progress

As stated before, the main goal of this research is to define a new model-driven methodology for WSN application development which allows developers to build more flexible and reusable designs. This goal was initially address considering the following sub-goals:

SG1. Define a WSN Domain-Specific Language (DSL) which enables the description of this kind of applications at a very high level of abstraction. Models built using this WSN DSL should pick up all functional and non-functional system requirements, but they should not include any design or implementation decisions. For this DSL to be really useful, a model editor (preferably a graphical one) must be implemented to support new Domain Model (DM) creation and validation.

SG2. Define the NesC [5] meta-model from the current language specification. A graphical modelling tool for creating new NesC component-based models would be desirable, but not required since NesC models will be automatically generated from WSN DSL ones.

SG3. Define a model-to-model (M2M) transformation which maps the concepts included in the WSN DSL to those included in the NesC meta-model.

SG4. Define a model-to-text (M2T) transformation which automatically generates NesC code from NesC models.

SG1 and SG2 were addressed in parallel by different teams. When these targets were completed (fully implemented and tested), part of the team working on the NesC meta-model started working on the M2T transformation (SG4), while the rest of the people involved in the research started working on the M2M transformation (SG3).

While the M2T transformation was successfully completed quite fast, the transformation between the WSN DSL and the NesC meta-model seemed a really hard problem to solve. This was mainly due to the huge semantic distance existing between the concepts included in both meta-models. At this point, two options were considered: (1) to introduce some lower abstraction concepts regarding system design into the DSL, or (2) to define an additional abstraction level between the two meta-models.

Option (1) meant reducing the abstraction level of the DSL and forcing developers to include design decision within their models. In contrast, option (2) offered evident advantages and just a few drawbacks. On the one hand, the use of an intermediate meta-model could help bridging the gap between WSN domain concepts and NesC primitives. As a consequence, and despite the need of defining two M2M transformations instead of one, the complexity of these transformations would be significantly lower. On the other hand, this intermediate meta-model could serve as an appropriate architecture description language for defining the system in terms of its components and the relationships existing between them.

With this aim in mind, a subset of the UML 2.0 [7] meta-model, including components (for describing the system structure) and state-machine and activity diagrams (for describing component behaviour), was defined to mediate between the WSN DSL and the NesC meta-model.

Fig. 1 (*left*) outlines the elements of the proposal, that is, the set of meta-models and model-transformations defined to obtain NesC code from WSN Domain Models (DM). The intermediate architecture description language has been highlighted in order to emphasize its key role in the process.

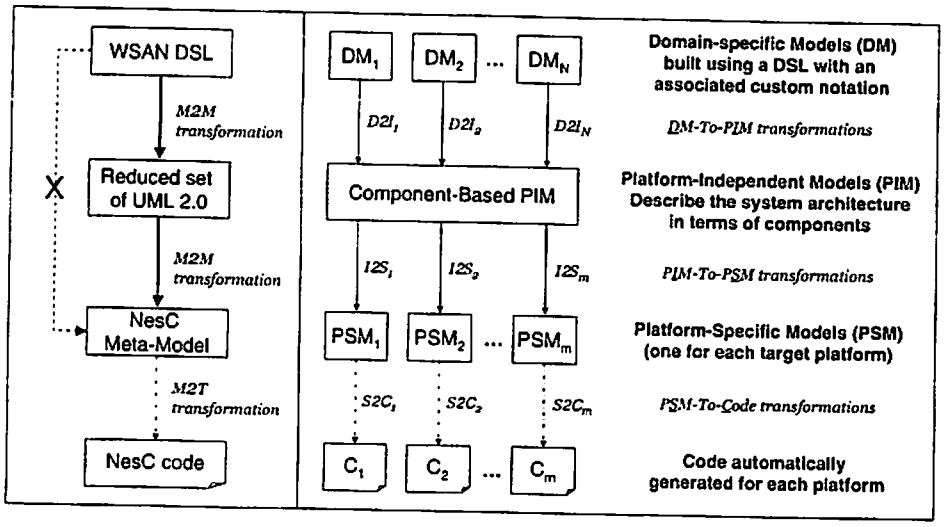

Fig. 1. (*Left*) Meta-models and model transformations included in the proposed MDE approach to WSN development. (*Right*) Models defined using the intermediate meta-model and the meta-model itself can be fully reused if new PSMs/DSLs were addressed in the future.

In addition to the already mentioned benefits of including an intermediate abstraction level, it is worth noting that models defined at this level can be fully reused if new target platforms were considered in the future. Furthermore, if new domains were addressed, the meta-model itself could be reused since it has been designed to be not only platform-independent but also domain-independent. This idea is illustrated in Fig. 1 (*right*).

2 The Eclipse Platform: The Selected MDE Environment

All the tools implemented as part of this work, including all the meta-models and model transformations outlined in the previous section, have been developed using the MDE facilities provided by the Eclipse platform. This free open-source environment offers one of the most widely used implementation of the OMG standard Meta-Object Facility (MOF) [8], called Eclipse Modelling Framework (EMF) [9].

Although EMF currently supports only a subset of MOF, called EMOF (Essential MOF), it allows designers to create, manipulate and store both models and meta-models. This is the reason why many MDE-related initiatives are currently being developed around Eclipse and EMF. Among them, and directly related to the tools implemented as part of this research, it is worth mentioning the following ones:

- The Graphical Modelling Framework (GMF) [10], which enables the implementation of graphical modelling tools from any EMF meta-model.
- The Eclipse Modelling Framework Technologies (EMFT) [11] which enables, among other things, the definition and evaluation of OCL queries and constraints on EMF models.
- The Atlas Transformation Language (ATL) [12], which provides the standard Eclipse solution for model-to-model transformations.
- MOFScript [13], which supports text (and more specifically code) generation from MOF-based models.

All these Eclipse plug-ins will be briefly detailed in the sections where the tools implemented as part of this work are presented.

3 Defining a WSN Domain-Specific Modelling Language

The definition of a WSN domain-specific modelling language (meta-model) is aimed at helping domain experts to describe their systems using only the WSN concepts they are familiar with. At this initial stage, no design decisions or concerns about the final target platform must be taken into account. Conversely, models at this level must provide a clear picture of system defined at a high level of abstraction. For instance, a WSN domain-specific model should supply information about the overall system functionality, how this functionality is partitioned into the different nodes, how this nodes are grouped and physically distributed, which information do they get from the environment, how do they communicate with each other and how frequently, where is the data processed/stored (locally or remotely), how it is presented to the user, etc.

The WSN Domain-Specific Language (DSL) presented in this paper has been designed to help domain experts to include all this information in their models. Thus, it provides the concepts most commonly used by the WSN community, together with the relationships that may appear between these concepts (see Fig. 2).

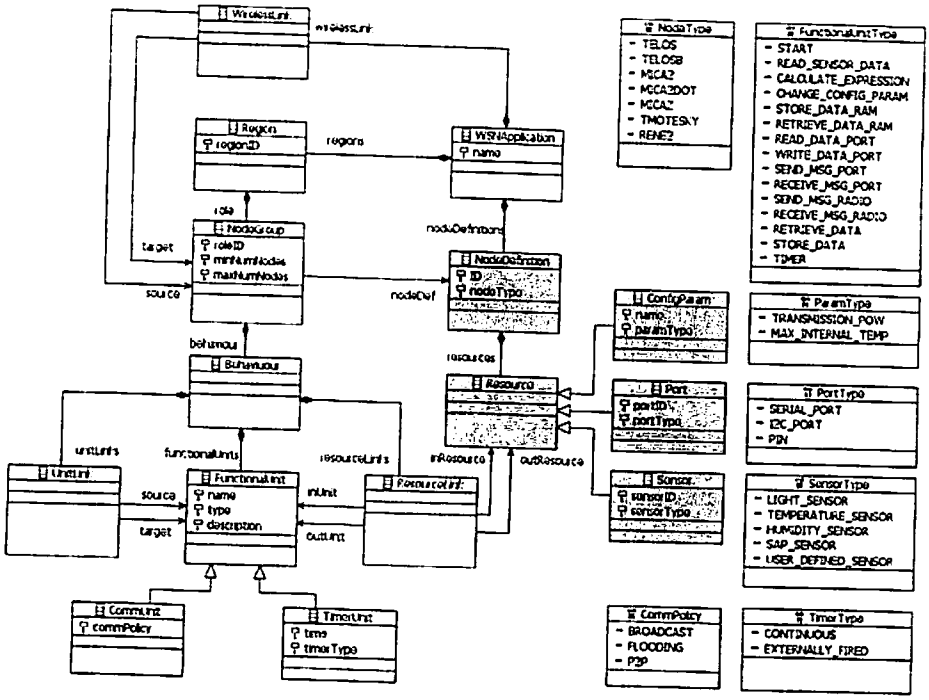

Fig. 2. WSN Domain Specific Language (WSN DSL)

Both structural and behavioural elements have been included in the meta-model. The structure of a WSN application is defined in terms of `Regions` connected by means of `WirelessLinks`. All the nodes performing similar tasks are grouped into a logical `NodeGroup`, while all the `NodeGroups` physically deployed together are considered to belong to the same `Region`.

The common `Behaviour` of the nodes belonging to the same `NodeGroup` is defined in terms of `FunctionalUnits`. The meta-model includes an enumerated set of predefined functional units (`FunctionalUnitType`) which contains, among others, data management (read/write from/to sensors/ports/memory), expressions calculation, timers, etc. `FunctionalUnits` can be linked together by means of `UnitLinks` and with external `Resources` (i.e. `Ports` and `Sensors`) by means of `ResourceLinks`. All these links together define the data-flow behaviour of the `NodeGroup`. Timers model explicitly internal node control-flow behaviour, while the external synchronization mechanisms regarding inter-node message passing is only implicitly represented in the models.

All the concepts and relationships included in this meta-model are quite useless if no tool is provided to support the creation and validation of new models from it. The following section presents the graphical notation and the modelling facility implemented on top of the WSN DSL previously described.

3.1 The WSN Graphical Modelling Tool

As previously stated, all the tools implemented as part of this work have been developed using the MDE facilities provided by the Eclipse Platform. In particular, the WSN DSL has been defined as an EMF [9] meta-model, and the graphical modelling tool, implemented to help domain experts to create new WSN models, has been developed using the GMF [10] Eclipse plug-in.

GMF allows designers: (1) to create a graphical representation for each domain concept appearing in a EMF meta-model, (2) to define a tool palette for creating and adding these graphical concepts to their models, and (3) to define a mapping between all the previous artefacts, i.e. meta-model concepts, their graphical representations, and the corresponding creation tools.

In addition, GMF can be used in conjunction with the EMFT [11] Eclipse plug-in to define new restrictions which can not be included in the meta-model (given the limitations of using class diagrams). These restrictions are defined as OCL constraints and they are validated at modelling time using the EMFT plug-in facilities.

The MITRA system, previously described in the introduction, has been depicted using the implemented WSN graphical modelling tool. As shown in Fig. 3, the model includes two regions. The first one contains two NodeGroups, one representing the MITRA nodes deployed in the almond orchard (SAP Monitoring NodeGroup) and another representing the Irrigation Control NodeGroup (containing only one node). The second region contains only one Sink NodeGroup with a single node. The behaviour of each of these NodeGroups has been defined following this three-step process:

1. Select the sensors to be read from those available in the selected NodeDefinition.
2. Select the activities performed by the NodeGroup from the enumerated set included in the WSN DSL, and
3. Link all these elements together to fulfil the system requirements, respecting the syntactic rules defined by the meta-model and the additional OCL rules included in the GMF application.

The following section describes how these WSN graphical models are automatically transformed into UML-based Platform-Independent Models (PIMs). Both the reduced set of the UML 2.0 meta-model, selected as the intermediate architecture description language, and the model-to-model transformation used to refine domain-specific models to the PIM level, are presented.

4 From Domain-Specific Models to UML-Based PIMs

As previously discussed in the introduction, we have chosen a simplified version of the standard UML 2.0 [7] meta-model as the intermediate specification language. This intermediate abstraction level is aimed at bridging the semantic gap existing between the domain and the platform meta-models, reducing the complexity of the required model transformations.

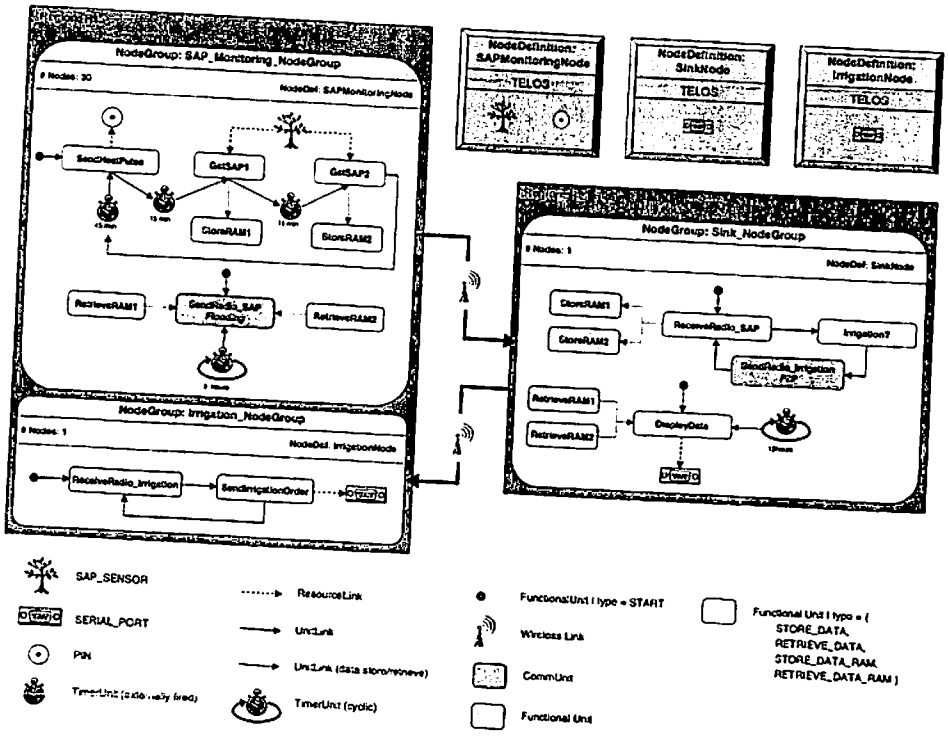

Fig. 3. MITRA system model depicted using the WSN DSL graphical modelling tool

The intermediate meta-model has been defined taking some of the elements included in the UML 2.0 meta-model and, more specifically, some of the concepts defined within the component, state-machine, and activity diagrams. Components are used to specify the system structure, while state-machines and activity diagrams are used to define component control-flow and data-flow behaviour, respectively.

The part of meta-model related to component specification includes simple and complex components, ports, port links, interfaces and services. The state-machine part includes states, pseudo-states (initial, join, and fork), orthogonal regions, transitions and events, and the activity diagram part, includes simple and complex activities (including conditionals and loops), timers, and activity links.

This meta-model has also been developed using EMF although, in this case, implementing a graphical modelling tool was unnecessary since (1) models at this level are not defined by the user but automatically obtained from DSL models and, (2) EMF provides a basic reflexive model editor which is sufficient to check the correctness of these intermediate models and which allows designers to manually introduce slight variations into them to test different architectural configurations.

Regarding the Model-to-Model (M2M) transformation required to refine DSL models into component-based PIMs, it has been implemented using the Eclipse Atlas Transformation Language (ATL) [12]. This hybrid declarative and imperative language enables to define mappings between the concepts included in different meta-models, that is, how each concept (or set of related concepts) in the source meta-model can be transformed into a concept (or set of related concepts) in the target

NesC meta-model is also simpler than the one offered by GRATIS II, which covers a wider range of TinyOS-based target platforms and configuration modes. However, our proposal is not TinyOS (or any other platform) dependent, enabling higher level WSN application designs. Currently we support TinyOS-NesC code generation like GRATIS II, although the proposal could be easily extended to different target platforms, as stated in the introduction (see Fig. 1).

The Abstract Task Graph (ATaG) [18] offers a DSL for graphically describing WSN applications in terms of the tasks they must perform and the data their nodes must collect. The data-driven diagrams depicted using this DSL provide a platform-independent model of the system under development (nodes, tasks, data types, etc. are abstract to keep this platform independence). These models are similar to the activity diagrams included in our intermediate component-based architectural models. However, ATaG is architecture-independent and thus, no design information can be included within its models. Furthermore, the ATaG code generation requires the user to provide the code of each abstract task included in the model for the current target platform, offering only a semi-automated solution.

Finally, CADENA [19] offers a very complete and sophisticated environment for general purpose application development. CADENA provides designers with an end-to-end model-driven environment which supports the entire application development life cycle. This tool offers, among others, a NesC plug-in which provides a graphical modelling tool (similar to the one offered by GRATIS II), and automatic NesC code generation facilities. WSN applications can also be modelled at a higher level of abstraction using CADENA general-purpose artefacts "adapted" to the WSN domain. However, adapting a general-purpose language or tool to a specific domain can be a hard work and the result could be never as good (in terms of precision, efficiency, etc.) as the one obtained by defining a DSL.

8 Conclusions and Future Research

The work presented in this paper offers a new model-driven approach to WSN application development. The proposal presents a high level of abstraction domain-specific language, which allows designers to model their systems in a platform-independent way, obtaining more flexible and reusable designs. Two additional abstraction levels have been defined which deal with the system architecture from a platform-independent and platform-specific point of view. Automatic model transformations from the initial domain-specific models to the final application code have been implemented using the Model-Driven Engineering facilities provided by the free open-source Eclipse platform. Both the proposed approach and the tools implemented to support it have been tested on a real WSN system related to *precision agriculture* with successful results. The re-engineered application is fully functional and, although it is not code-optimized, the effort and the time-to-market to produce the new solution have been sensibly reduced. Actually, different solutions have been effortlessly generated thanks to the implemented infrastructure, allowing us to test new sensors and different network topologies and communication protocols.

Currently, we are working on improving the NesC meta-model and the ATL transformation from PIMs to NesC models in order to keep all the behavioural

information during the whole development process. This will allow us to simplify the final code generation step using only NesC models as input. We are also working on the integration of requirements from the very early stages of the proposed MDE approach. Actually, we have developed a Requirements Engineering Meta-Model (REMM) and a graphical requirements modelling tool aimed at defining reusable requirements catalogues [20]. Currently we are building a catalogue of functional and non-functional WSN requirements together with a tracing tool. We are also very interested in proving the benefits of our intermediate meta-model for different target platforms and domain applications. Thus, we plan to define new DSLs for other domains in which our research team has also some experience, such as computer vision and robotics. The wide variety of platforms currently available for this kind of systems, and the fact that some applications incorporate concepts from both domains (i.e. industrial inspection), make this future research both challenging and promising.

References

1. Akyildiz, I.F., Kasimoglu, I.H.: Wireless Sensor and Actor Networks: research challenges. Ad Hoc Networks 2, 351–367 (2004)
2. Römer, K., Mattern, F.: The design space of wireless sensor networks. IEEE Wireless Communications 11, 54–61 (2004)
3. Huang, G.T.: Casting the Wireless Sensor Net. MIT's Magazine of Innovation, 51–56 (2003)
4. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System Architecture Directions for Networked Sensors, vol. 34. ACM Press, New York (2000)
5. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesC Language: A Holistic Approach to Network Embedded Systems. In: ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, San Diego, California, USA, pp. 1–11 (2003)
6. Kent, S.: Model Driven Engineering. In: Butler, M., Petre, L., Sere, K. (eds.) IFM 2002. LNCS, vol. 2335, Springer, Heidelberg (2002)
7. Unified Modeling Language: Superstructure v 2.0. The Object Management Group (2005)
8. Meta-Object Facility Specification v2.0: The Object Management Group (2004)
9. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modelling Framework. Addison-Wesley Professional, Reading (2003)
10. The Eclipse Graphical Modelling Framework, available at: http://www.eclipse.org/gmf
11. The Eclipse Modedelling Framework Technologies (EMFT) Projects, available at: http://www.eclipse.org/emft/projects/
12. The Atlas Transformation Language (ATL) Project, available at: http://www.eclipse.org/m2m/atl/
13. The Eclipse MOFScript subproject, available at: http://www.eclipse.org/gmt/mofscript/
14. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: An Acqusitional Query Processing System for Sensor Networks. ACM Transactions on Database Systems 30, 122–173 (2005)
15. Marrón, P.J., Minder, D., Lachenmann, A., Rothermel, K.: TinyCubus: An Adaptive Cross-Layer Framework for Sensor Networks. Information Technology 47, 87–97 (2005)
16. GRATIS II: Institute for Software Integrated Systems. Vanderbilt University, Tennessee, USA, available at: http://www.isis.vanderbilt.edu/projects/nest/gratis

17. Lédeczi, Á., Maróti, M., Völgyesi, P.: The Generic Modeling Environment (GME). Institute for Software Integrated Systems, Vanderbilt University, Tennessee, USA, available at: http://www.isis.vanderbilt.edu/Projects/gme
18. Bakshi, A., Prasanna, V.K., Reich, J., Larner, D.: The Abstract Task Graph: A Methodology for Architecture-Independent Programming of Networked Sensor Systems. In: EESR'05. Proc. Workshop on End-to-End, Sense-and-Respond systems, applications and services, Seattle, Washington, pp. 19–24 (2005)
19. The Cadena 2.0 Project: Kansas State University, USA, available at: http://cadena.projects.cis.ksu.edu/
20. Vicente-Chicote, C., Moros, B., Toval, A.: REMM-Studio: an Integrated Model-Driven Environment for Requirements Specification, Validation and Formatting. Journal of Object Technology, Special Issue TOOLS EUROPE 2007 (2007)

Appendix A: Excerpt of the MOFScript M2T Transformation

This excerpt of the implemented MOFScript transformation is the module in charge of generating the code for all the NesC interfaces defined in NesC models.

```
texttransformation NesCM2T (in mm:"NesC") {
...
module::createInterfaces(){
   self.objectsOfType(mm.Interface)->forEach(i:mm.Interface){
      file f ( i.name + ".nc" )
      f.println ( "interface " + i.name + "{" )
      var count:integer
      // Adds the interface command prototypes
      i.prototypes->forEach( p:mm.CommandPrototype ){
         if ( p.isAsynchronous==true ) f.print ("async command ")
         else f.print ( "command " )
         f.print ( p.returnType + " " + p.name + "(" )
         count = p.arguments.size()
         p.arguments->forEach ( v:mm.Variable) {
            f.print ( v.type + " " + v.name )
            if ( count > 1 ) {f.print (", ") count=count-1}
         } f.println ( ");" )
      } // Event prototypes are similarly added ...
}}
```

Appendix B: Excerpt of the Generated NesC Code

This excerpt of the generated NesC code corresponds to the MitraM module defined in the NesC component model (see Fig. 5).

```
includes MitraMsg;
module MitraM {
   uses {
      interface StdControl as CommControl;
      interface StdControl as TimerControl;
      interface StdControl as ADCControl;
```

```
    interface SendMsg as Send;
    interface ADC;
    interface Timer as Timer1;
    interface Timer as Timer2;
    interface Timer as Timer3;
    interface Timer as Timer4; }
  provides {
    interface StdControl; }
}
implementation {
  TOS_Msg msg_global;
  uint8_t counter=1;
  uint16_t reading1, reading2;
  command result_t StdControl.start() {
    call ADCControl.start();
    call TimerControl.start();
    call CommControl.start();
    call Timer4.start(TIMER_REPEAT, 1024*60*60*3);
    initialize_cycle();
    return SUCCESS; }
  ...
  event result_t Timer1.fired() {
    SENSOR_SEND_PULSE();
    call Timer2.start(TIMER_ONE_SHOT, 1024*60*15);
    return SUCCESS;}
  ...
  event result_t Timer4.fired(){
    // Builds a message containing sap measures and sends it to the sink node via wireless.
    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;}
}
```

meta-model. One-to-one transformations are desirable but commonly they are only possible when the concepts included in the two meta-models are semantically close.

In this case, the transformation from the WSN DSL and the intermediate UML-based meta-model requires some relatively complex mappings, although some of them are also quite direct. Some of the rules, included in this ATL model transformation, are outlined next:

- Each `NodeGroups` is mapped to a `Component`.
- Given the data-flow oriented behaviour of WSN `NodeGroups`, a very simple `StateMachine` is associated to each `Component` including only an `Initial PseudoState`, and two `States`: *Working* and `Final`.
- Uncoupled sets of `ActivityUnits` are placed into different `OrthogonalRegions` in the *Working* `State`, since these activities are executed in parallel. This is the most complex transformation rule since it requires detecting unconnected graphs of activities.
- Each `Timer FunctionalUnit` is mapped to a UML `TimerActivity`.
- All the messages sent via wireless from a `NodeGroup` to another are modelled as UML `SendSignalActions`.
- Signals send by each NodeGroup to an output resource (`Ports`) are converted into UML `SendSignalActions` and, conversely, signals received from input resources (`Sensors`) are transformed into UML `AcceptEventActions`.
- Each `AcceptEventAction` requires adding a new `Event` to the `StateMachine` and an `InternalTransition` in the *Working* `State` fired by this event.

The result of applying the ATL transformation on the initial MITRA DSL model is a UML-like platform-independent component model, which describes the system structure and behaviour in terms of its components and the relationships existing between them. Although, as stated before, we have not implemented a graphical model editor for this intermediate level, the following figure (created using a basic

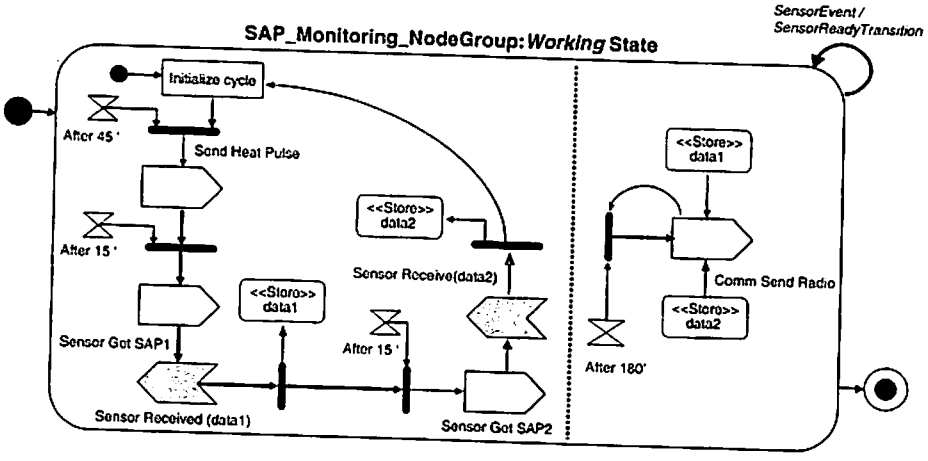

Fig. 4. Behaviour of the SAP monitoring `Component`

drawing tool) has been depicted using the UML graphical notation in order to provide an easier to understand representation of part of the resulting model, and more precisely, the behaviour of the SAP Monitoring `Component` (`NodeGroup`).

As it can be readily appreciated in Fig. 4, the transformation has divided the *Working* `State` of the SAP Monitoring `Component` into two `OrthogonalRegions`. These regions contain the two uncoupled sets of activities identified in the `NodeGroup` (see Fig. 3), one describing the sensing loop and another describing how the collected data is sent to the Sink `NodeGroup` via wireless. Similarly, the *Working* `State` of the Sink `Component` is also divided into two `OrthogonalRegions`, one including the activities related to data collection (from SAP monitoring nodes), irrigation need estimation, and control message delivery (to the Irrigation Control NodeGroup), and another including data retrieval and display activities.

The following section describes how these intermediate models are automatically transformed into TinyOS-NesC Platform-Specific Models (PSMs), applying a new ATL model-to-model transformation.

5 From PIMs to NesC Component Models

Nowadays, TinyOS [4] is the most widely used operating system for WSN application development and, accordingly, a wide variety of tools supporting it can be currently found in the marketplace. Among them, the solution developed by the TinyOS team is the NesC [5] component-based programming language, also extensively used.

A NesC meta-model has been implemented, according to the NesC 1.1 specification, to support the last stage of the proposed MDE approach. This meta-model comprises the following concepts: `Modules` (which define component implementation), `Configurations` (which define component groups and `Wirings` between them), and `Interfaces` (which include a list of `CommandPrototypes` and `EventPrototypes`). Modules must implement all the `Commands` included in the interfaces they provide and all the `Events` in the interfaces they use.

NesC applications are designed following a quite regular component pattern and thus, the rules needed to define the ATL transformation between PIMs and NesC models are not very complex. Some of these rules are outlined next:

- A different NesC model is created for each `Component` in the PIM.
- All `TimerActivities` defined in the PIM are implemented by a unique predefined NesC module, called *TimerC*. This module provides a parameterized *Timer* interface which includes a *fired* event.
- Conversely, all `SendSignalActions` are implemented by a unique predefined *GenericComm* module which provides different interfaces for each type of message. All these interfaces include a *SendMsg* event.
- A Main module must be necessarily included in the design since it is invoked when the application starts. This module is in charge of initializing the rest of modules.
- A top level `Configuration` module must be defined to wire all the previously defined components.

Fig. 5 presents the NesC model associated to the sap monitoring component defined in the PIM. Like in the previous case, this graphical representation has been manually depicted using a conventional drawing tool, since a graphical NesC model editor has not been implemented yet. However, the depicted model faithfully represents the model obtained as the result of applying the implemented ATL transformation.

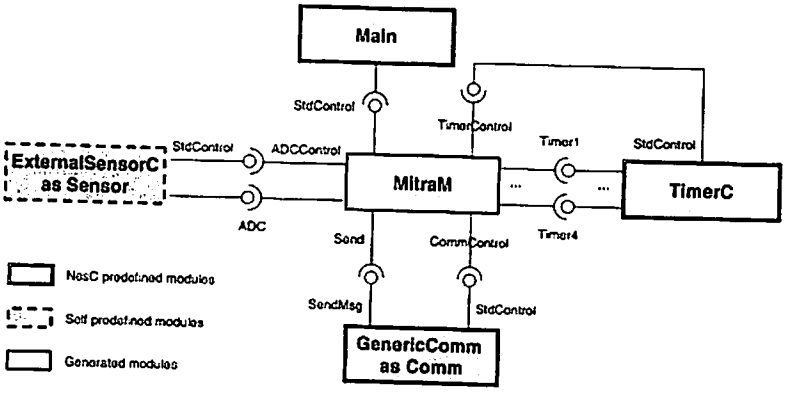

Fig. 5. NesC component model for the sap monitoring nodes

NesC models, described according to the implemented NesC meta-model, only enable the description of components and their interconnections, and do not include any of the behavioural information described in the more abstract models. As explained in the following section, this requires using NesC models and PIMs in order to generate the final application code. Some improvements in the NesC meta-model and in the ATL transformation from PIMs to NesC models are currently being developed to address this limitation, as later commented in the conclusions.

6 Code Generation

The final step of the proposed MDE methodology for WSN system development is to obtain the final application code from the previous models. In this case, the transformation is not Model-to-Model (M2M) but Model-to-Text (M2T). M2T transformations define how model elements are converted into text patterns (in this particular case, into code), while M2M transformations define meta-model mappings, that is, how the elements (concepts and relationships between them) included in one of the meta-models are transformed into elements included in the other.

The final NesC M2T transformation has been implemented using the Eclipse MOFScript [13] plug-in. This tool enables the definition of M2T transformations for MOF-based models. It provides, among others, the following facilities: model checking, parsing and querying, output file management, code completion, etc.

In order to obtain the final application code, the NesC models obtained in the previous step are not sufficient, since they do not contain complete information about component behaviour. Thus, the MOFScript transformation has been designed to use

both, NesC models and intermediate PIM models (containing component behaviour specifications obtained from the initial DSL models). This limitation is currently being addressed as described in the conclusions and future research section.

Some of the rules included in the MOFScript transformation, are outlined next:

- The NesC component model enables the generation of a list of provided and required interfaces (`provides` and `uses` clauses, respectively) included at the beginning of the NesC file.
- The module implementation starts with a variable declaration section. One variable is defined for each `<<Store>> Activity` defined in the PIM with a different name, and also for each message the `Component` sends (receives) to (from) other components via wireless.
- For each provided `Interface` in the NesC model, all its `Commands` must be implemented. Conversely, for each required `Interface`, all its `Events` must be handled. The sequence of NesC primitives associated to these `Commands` and `Events` is extracted from the `Activities` defined in the PIM.

An excerpt of the implemented MOFScript transformation can be found in Appendix A, and a piece of the generated NesC code in Appendix B, both of them included at the end of the paper. The generated solution, obtained from the corresponding PIM intermediate model (see Fig. 4) and the NesC component model (see Fig. 5), has been successfully compiled and deployed on the MITRA WSN system, demonstrating satisfactory results.

7 Related Work

Different approaches to WSN application development can be found in the literature. Some of them offer a set of predefined components, built on top of a certain operative system, and allow designers to build new ones by configuring and combining them. This is the case of TinyDB [14], which defines a database of TinyOS [4] components that can be distributed and run in different nodes of a WSN. The information obtained by each node can be accessed from an external PC by means of SQL-like queries. Also in this line, TinyCubus [15] offers a framework which enables to dynamically select and interconnect predefined TinyOS components. These two proposals, as most others, are totally focused on the implementation of platform-specific WSN applications. These approaches require a deep knowledge of the target platform and, in most cases, the resulting designs are too platform-dependent to be reused.

Trying to address this problem, and in the line of this paper, some proposals have focused their attention on the model-driven approach. GRATIS II [16] offers a graphical modelling tool for designing component-based NesC applications. The underlying NesC meta-model has been defined using the Generic Modelling Environment (GME) [17], a toolkit for defining domain-specific modelling and programming languages. GRATIS II offers a solution similar to the one offered by the final step of our proposal, that is, the one including our NesC meta-model and NesC model-to-code transformation. Currently we do not offer a graphical modelling tool for depicting NesC component models since these are automatically generated from the higher level meta-model, and not depicted by the user like in GRATIS II. Our

NesC meta-model is also simpler than the one offered by GRATIS II, which covers a wider range of TinyOS-based target platforms and configuration modes. However, our proposal is not TinyOS (or any other platform) dependent, enabling higher level WSN application designs. Currently we support TinyOS-NesC code generation like GRATIS II, although the proposal could be easily extended to different target platforms, as stated in the introduction (see Fig. 1).

The Abstract Task Graph (ATaG) [18] offers a DSL for graphically describing WSN applications in terms of the tasks they must perform and the data their nodes must collect. The data-driven diagrams depicted using this DSL provide a platform-independent model of the system under development (nodes, tasks, data types, etc. are abstract to keep this platform independence). These models are similar to the activity diagrams included in our intermediate component-based architectural models. However, ATaG is architecture-independent and thus, no design information can be included within its models. Furthermore, the ATaG code generation requires the user to provide the code of each abstract task included in the model for the current target platform, offering only a semi-automated solution.

Finally, CADENA [19] offers a very complete and sophisticated environment for general purpose application development. CADENA provides designers with an end-to-end model-driven environment which supports the entire application development life cycle. This tool offers, among others, a NesC plug-in which provides a graphical modelling tool (similar to the one offered by GRATIS II), and automatic NesC code generation facilities. WSN applications can also be modelled at a higher level of abstraction using CADENA general-purpose artefacts "adapted" to the WSN domain. However, adapting a general-purpose language or tool to a specific domain can be a hard work and the result could be never as good (in terms of precision, efficiency, etc.) as the one obtained by defining a DSL.

8 Conclusions and Future Research

The work presented in this paper offers a new model-driven approach to WSN application development. The proposal presents a high level of abstraction domain-specific language, which allows designers to model their systems in a platform-independent way, obtaining more flexible and reusable designs. Two additional abstraction levels have been defined which deal with the system architecture from a platform-independent and platform-specific point of view. Automatic model transformations from the initial domain-specific models to the final application code have been implemented using the Model-Driven Engineering facilities provided by the free open-source Eclipse platform. Both the proposed approach and the tools implemented to support it have been tested on a real WSN system related to *precision agriculture* with successful results. The re-engineered application is fully functional and, although it is not code-optimized, the effort and the time-to-market to produce the new solution have been sensibly reduced. Actually, different solutions have been effortlessly generated thanks to the implemented infrastructure, allowing us to test new sensors and different network topologies and communication protocols.

Currently, we are working on improving the NesC meta-model and the ATL transformation from PIMs to NesC models in order to keep all the behavioural

information during the whole development process. This will allow us to simplify the final code generation step using only NesC models as input. We are also working on the integration of requirements from the very early stages of the proposed MDE approach. Actually, we have developed a Requirements Engineering Meta-Model (REMM) and a graphical requirements modelling tool aimed at defining reusable requirements catalogues [20]. Currently we are building a catalogue of functional and non-functional WSN requirements together with a tracing tool. We are also very interested in proving the benefits of our intermediate meta-model for different target platforms and domain applications. Thus, we plan to define new DSLs for other domains in which our research team has also some experience, such as computer vision and robotics. The wide variety of platforms currently available for this kind of systems, and the fact that some applications incorporate concepts from both domains (i.e. industrial inspection), make this future research both challenging and promising.

References

1. Akyildiz, I.F., Kasimoglu, I.H.: Wireless Sensor and Actor Networks: research challenges. Ad Hoc Networks 2, 351–367 (2004)
2. Römer, K., Mattern, F.: The design space of wireless sensor networks. IEEE Wireless Communications 11, 54–61 (2004)
3. Huang, G.T.: Casting the Wireless Sensor Net. MIT's Magazine of Innovation, 51–56 (2003)
4. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System Architecture Directions for Networked Sensors, vol. 34. ACM Press, New York (2000)
5. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesC Language: A Holistic Approach to Network Embedded Systems. In: ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, San Diego, California, USA, pp. 1–11 (2003)
6. Kent, S.: Model Driven Engineering. In: Butler, M., Petre, L., Sere, K. (eds.) IFM 2002. LNCS, vol. 2335, Springer, Heidelberg (2002)
7. Unified Modeling Language: Superstructure v 2.0. The Object Management Group (2005)
8. Meta-Object Facility Specification v2.0: The Object Management Group (2004)
9. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modelling Framework. Addison-Wesley Professional, Reading (2003)
10. The Eclipse Graphical Modelling Framework, available at: http://www.eclipse.org/gmf
11. The Eclipse Modedelling Framework Technologies (EMFT) Projects, available at: http://www.eclipse.org/emft/projects/
12. The Atlas Transformation Language (ATL) Project, available at: http://www.eclipse.org/m2m/atl/
13. The Eclipse MOFScript subproject, available at: http://www.eclipse.org/gmt/mofscript/
14. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: An Acqusitional Query Processing System for Sensor Networks. ACM Transactions on Database Systems 30, 122–173 (2005)
15. Marrón, P.J., Minder, D., Lachenmann, A., Rothermel, K.: TinyCubus: An Adaptive Cross-Layer Framework for Sensor Networks. Information Technology 47, 87–97 (2005)
16. GRATIS II: Institute for Software Integrated Systems. Vanderbilt University, Tennessee, USA, available at: http://www.isis.vanderbilt.edu/projects/nest/gratis

17. Lédeczi, Á., Maróti, M., Völgyesi, P.: The Generic Modeling Environment (GME). Institute for Software Integrated Systems, Vanderbilt University, Tennessee, USA, available at: http://www.isis.vanderbilt.edu/Projects/gme
18. Bakshi, A., Prasanna, V.K., Reich, J., Larner, D.: The Abstract Task Graph: A Methodology for Architecture-Independent Programming of Networked Sensor Systems. In: EESR'05. Proc. Workshop on End-to-End, Sense-and-Respond systems, applications and services, Seattle, Washington, pp. 19–24 (2005)
19. The Cadena 2.0 Project: Kansas State University, USA, available at: http://cadena.projects.cis.ksu.edu/
20. Vicente-Chicote, C., Moros, B., Toval, A.: REMM-Studio: an Integrated Model-Driven Environment for Requirements Specification, Validation and Formatting. Journal of Object Technology, Special Issue TOOLS EUROPE 2007 (2007)

Appendix A: Excerpt of the MOFScript M2T Transformation

This excerpt of the implemented MOFScript transformation is the module in charge of generating the code for all the NesC interfaces defined in NesC models.

```
texttransformation NesCM2T (in mm:"NesC") {
...
module::createInterfaces(){
   self.objectsOfType(mm.Interface)->forEach(i:mm.Interface){
      file f ( i.name + ".nc" )
      f.println ( "interface " + i.name + "{" )
      var count:integer
      // Adds the interface command prototypes
      i.prototypes->forEach( p:mm.CommandPrototype ){
         if ( p.isAsynchronous==true ) f.print ("async command ")
         else f.print ( "command " )
         f.print ( p.returnType + " " + p.name + "(" )
         count = p.arguments.size()
         p.arguments->forEach ( v:mm.Variable) {
            f.print ( v.type + " " + v.name )
            if ( count > 1 ) {f.print (", ") count=count-1}
         } f.println ( ");" )
      } // Event prototypes are similarly added ...
}}
```

Appendix B: Excerpt of the Generated NesC Code

This excerpt of the generated NesC code corresponds to the MitraM module defined in the NesC component model (see Fig. 5).

```
includes MitraMsg;
module MitraM {
   uses {
      interface StdControl as CommControl;
      interface StdControl as TimerControl;
      interface StdControl as ADCControl;
```

```
    interface SendMsg as Send;
    interface ADC;
    interface Timer as Timer1;
    interface Timer as Timer2;
    interface Timer as Timer3;
    interface Timer as Timer4; }
  provides {
    interface StdControl; }
}
implementation {
  TOS_Msg msg_global;
  uint8_t counter=1;
  uint16_t reading1, reading2;
  command result_t StdControl.start() {
    call ADCControl.start();
    call TimerControl.start();
    call CommControl.start();
    call Timer4.start(TIMER_REPEAT, 1024*60*60*3);
    initialize_cycle();
    return SUCCESS; }
  ...
  event result_t Timer1.fired() {
    SENSOR_SEND_PULSE();
    call Timer2.start(TIMER_ONE_SHOT, 1024*60*15);
    return SUCCESS;}
  ...
  event result_t Timer4.fired(){
    // Builds a message containing sap measures and sends it to the sink node via wireless.
    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;}
}
```