A Model Driven Architecture Framework for Robot Design and Automatic Code Generation

Anne-Lise Courbis, Kahune Luu, Benjamin Grondin, Kelly Roussel

A Model Driven Architecture Framework for Robot Design and Automatic Code Generation

Anne-Lise Courbis
LGI2P
IMT Mines Ales, Univ. Montpellier
Alès, France
anne-lise.courbis@mines-ales.fr

Benjamin Grondin
Department of Informatics & Artificial Intelligence
IMT Mines Ales, Univ. Montpellier
Alès, France
benjamin.grondin@mines-ales.org

Kahune Luu
Department of Informatics & Artificial Intelligence
IMT Mines Ales, Univ. Montpellier
Alès, France
kahune.luu@mines-ales.org

Kelly Roussel
Department of Informatics & Artificial Intelligence
IMT Mines Ales, Univ. Montpellier
Alès, France
kelly.roussel@mines-ales.org

***Abstract*— This work presents a research and development experiment in software engineering at the IMT Mines Ales, France. The goal is to define a framework allowing a system controller to be graphically designed and its java code to be automatically generated. This framework is expected to be a support for students following the system engineering curriculum, and who have to program LEGO Mindstorms EV3 robots although they have not already been trained to concurrent Java programming. The experimental methodology focuses on learning and implementing the following paradigms: model driven design, software architecture for event driven systems and reactive system programming using JAVA threads. We present the design framework defined during this experiment, and the feedback of students who have been involved in setting up the state of the art and developing the framework.**

Keywords—model driven architecture, embedded system modelling, system engineering process, state machine modelling, LEGO Mindstorms EV3, JAVA programming

I. INTRODUCTION

Designing an embedded software for a robot whose mission requires both remote communications and the recognition of its environment requires skills about system engineering, architecture design, behaviour modelling, concurrent programming and safety property verification. Such concepts are difficult to learn for students who are beginners in software or system engineering. The difficulties come from theoretical concepts and practical ones as well. The main theoretical concepts are multi component system engineering, design methods and more precisely the model driven architecture (MDA) approach, concurrency paradigms of the software part and system verification to detect for example deadlocks. Practical difficulties concern the use of appropriate tools to design such systems starting from their requirement until their implementation.

An experiment named research & development mission (R&D Mission) has been conducted at IMT Mines Alès, France. This pedagogical exercise aims at sensitizing students to research activities through two parts: the study of the state of the art and the development of a research issue conducted by members of the laboratory supporting the departement curriculum. The students follow a generalist curriculum of engineering during the first year. They have been involved for six months in a software engineering department named Informatics and Artifical Intelligence. Until now, their knowledge was focusing on the basis of JAVA programming.

The goal of this experiment is to involve students in defining and implementing:

- a graphical framework called RMA (robot model-driven architecture) for designing the configuration of a LEGO Mindstorms robot [1] and describing their behaviour from a high level of abstraction and,
- a JAVA code module called RMDA2JAVA for generating the architecture of the software that will be embedded in the EV3 module of the robot. In this first experiment, the architecture is a JAVA code skeleton decomposed into packages and classes that has to be completed.

There are several reasons for conducting this experiment on LEGO Mindstorms EV3 robot. First, it is an actual motivation for students to discover robot progamming by using a *learn and play* method as a means to learn strong concepts. Furthermore, they feel helpful and responsible because the developed framework is expected to be used by other students belonging to System Engineering (SE) Departement. Indeed, students of the SE Departement are following a course about complex system design and are involved in a project consisting in developing and assembling a robot following the methods and concepts of the AFIS (French Association for SE) community [2], which is the French part of the INCOSE [3]. Some of them participate to the French contest ROBAFIS [4], organized by the AFIS. Students of the SE Department have few skills in programming, and are not motivated by JAVA programming. Using a framework developed by other students may be a real motivation to learn advanced concepts in Java and develop suitable controller to be embedded in their robot.

The paper is divided in five sections, including this introduction. Section two concerns the study of existing works about Lego Mindstorms programming using MDA approach. Section three introduces conceptual requirements that are fundamental for the framework definition and implementation. Furthermore, these concepts are required for students' training in software engineering. Section four concerns the implementation of the framework: it points out how conceptual requirements have been applied by students for developing the RMDA Framework and what are the results of this work. An example will illustrate this part in order to highlight concepts, methods and results. Section five

will close the article by a discussion about the feedback of students about this experiment and the strength and weakness of this pedagogical method. We will also present our perspectives about the enhancement of the framework.

II. Related works

The first part of the R&D Mission performed by students has consisted in setting up a state of the art of the issue expressed in the previous part. The time of this part being quite short, the subject was precise: the use of model-driven architecture approach for programming LEGO Mindstorms robots (more precisely the controller part, called EV3 brick), or eventually other kind of robots, versus conventional programming techniques such as procedural or object-oriented programming. The graphical environment of programming using preset blocks [5] proposed by LEGO Mindstorms and powered by LabView has been considered by students, but qualified as irrelevant for applying MDA approach.

Before starting the study, non familiar concepts have been defined. Model-Driven Engineering (MDE) is an approach for software development whereby models are used as the primary source for documenting, analyzing, designing, constructing, deploying and maintaining a system [6].

Model-Driven Architecture (MDA): MDA itself is not a new OMG (Object Management Group) specification but rather an approach to software development which is enabled by existing OMG specifications [6]. In the MDA approach, models are the key elements for expressing user requirements, designing the target system, documenting its development, testing and deployment, and finally, enhancing and upgrading systems.

First feedback of this state of the art is that the model-driven architecture approach is not such widespread in robotics controller programming than the object oriented programming. The study began by searching if such approach had ever been used in LEGO Mindstorms Robot programming. LEGO Mindstorms Robots have been considered for several years as a means to support student training [7], [8]. In that context, robots are used for education and unfortunately, MDA approach and independence regarding technological evolution is not considered as a priority in tutorials [1]. Different approaches can be considered [11], such as functions, objects, modules, agents or components ones.

Despite many works concerning Mindstorms programming for education purpose, we could identify few refererences using any model driven. For example, there is a modelling environment dedicated to EV3 programming [9] developed by Obeo Designer [10] that offers some facilities to model robot choregraphy and generate JAVA code, but from our point of view, this approach mixes concepts of software engineering that will be explicitely presented in the next section and does not teach to *think system*. It is a nice way for introducing programming but not enough powerful to learn good practices to future engineers.

Nevertheless, some works are interesting and offer advice for system architecting and programming. For example, [8] reveals the pros of developing a platform of control for the LEGO Mindstorms Robot. Various types of software architectures are presented in [11], such as deliberative architecture, subsumption architecture [12] or hybrid architecture. It also presents the principle of an architecture with two layers, used in robotics.

We have also studied some references about MDE. Reference [13] has to be used as a guide considering the details given about the key points for a good functional component-oriented program. In the same way, [14] deals with the application of model-driven engineering to service robots step by step from Unified Modelling Language (UML) model to dynamic architecture. The document [15] describes the structure of a component-oriented program and the structure of a model-driven software development. The latter relies on a strong PIM (Platform Independent Model).

Developing a real-time embedded system relies mostly on concurrent programming. The lesson in [16] has been useful for learning the multi-threading concept and controlling concurrency between threads.

We studied also some MDE approach applied to robotics. Both [17] and [18] detail the code-generation from the model established and introduce tools to do so. Compared to the few documents quoted above, [19] brings to light a particular model-to-text environment, called RobotML. As suggested in this name, it uses UML diagrams to generate a code. RobotML is the closest related work we have found. Our study differentiates from it since LEGO Mindstorms Robot EV3 is specifically concerned. Another study is presented in [20] pointing out the interest of MDA. However, this work does not integrate the architecture design using a specific domain language and its associated graphical framework.

Finally, we inspired our work from [19], introducing RobotML as a Domain-Specific language for robotics application as well as the related work of [21] which focused on Aibo robot to develop the same kind of approach we want to implement for Lego MindStorms Robots.

Two previous R&D Missions, developed during the two last years by students of IMT Mines Alès, have also been studied for conducting this experiment: [22] is about the definition of robot behavioural patterns expressed in eFFBD (Enhanced Functional Flow Block Diagram) and their transformation in state machines; the other one [23] is about the implementation of a EV3 controller modelled by a state machine in JAVA using the LeJOS library [24].

III. Conceptual Requirements for defining and implementing the framework RMDA

To develop a component oriented program for coding LEGO Mindstorms robots, we will carefully follow the five requirements stated by [13] to avoid architectural erosion. Moreover, our work will be guided by the work of [14] [26] to make sure we apply the principle of separation of concerns and use patterns to manage concurrency. As defined in [15], our work focuses on the PIM layer in order to remain independent of the technological platform used for the software implementation. A PIM is a formal specification of the structure and function of a system that abstracts away technical detail [26]. Essential goal of MDA is to derive value from models and modelling that helps us deal with the complexity and interdependence of complex systems [27].

This approach presents many advantages as flexible adaptation to technological evolution, a larger possibility to reuse parts of the code. This last point is one of the main aim of our work in order to help developers in coding the robot

not having to write the whole program each time a change of component is made on the robot.

We focus on MDA through the two following paradigms:

- The separation of concerns and model abstraction,
- The meta modelling and model transformation.

Another paradigm will be presented, since it has been an actual guideline to define the generic architecture of the robot and has implied a generic architecture of the controller:

- The Systemic modelling approach [28].

Lastly, the choice for representing the controller according to an abstract view has been oriented by work of the research team who has proposed this R&D mission [25][29]. Indeed, the next step of this project will be to verify the robot liveness and safety properties through formal approaches, based on its modelling before statting its implementation. For this reason, one important modelling choice is:

- The representation of the controller by a state machine

We develop these paradigms in the above sections and for each of them, we point out how it has been applied in the project.

A. *Separation of Concerns and Abstraction*

MDA helps support separation of concerns by enabling different viewpoints of a system as well as the transformation between levels of abstraction. It is defined as the essential characteristics of quality architecture: when concerns are separated it is possible to deal with, understand and specify one aspect of a system without undue dependencies on other aspects. Separation of concerns enables greater agility, ability to deal with change and a "divide and conquer" approach to realize a system [27].

Abstraction deals with the concepts of understanding a system in a more general way; said in more operational terms, with abstraction one eliminates certain elements from the defined scope; this may result in introducing a higher level viewpoint at the expense of removing detail [27]. Abstraction is a powerful way of modelling since details are postpone to next steps, allowing designer to concentrate on main finalities of the system. Furthermore, abstraction does not eliminate the possibility to evaluate some fundamental properties such as liveness and safety.

In our project, separation of concerns is achieved by defining an architecture that comes from the SE processes applied by students from the SE Department. Processes implied to define models and verify both their consistency and their traceability all along the project development through the following steps: user requirements, system requirements, functional view of the system, logical view, and finally, the physical view.

In our project, separation of concerns and abstraction are achieved by applying the following design rules:

- The physical view is designed at a high level as a general pattern for any robot design. This view consists in setting up high-level abstraction models corresponding to the main sub-systems of the robot. These sub-systems have been defined by considering Systemic Modelling which is presented in the next sub-section.
- Each part of the system (component, subsystems) is seen by its environement as a black box with provided and required interfaces. Information is accessible via ports which deliver or provide services [29].
- Models are broken down in sub-models. These refinement steps may be validated using formal methods [25] that ensures the preservation of liveness and safety properties.

This modelling approach includes concepts to represent services and components as primary elements in the robotics system. As stated in [26], main features for quality of architectures are high cohesion and loose coupling which leads to design for change. This is achieved by implementing the principles of abstraction, separation of concerns, and information hiding.

B. *Systemic Modelling*

Systemic Modelling has been introduced in France by J.L. Le Moigne[28][30]. His theory comes from two sources: the structuralism theory from Piaget's and the cybernetic theory of N. Wiener. His initial work focused on Information System, but he developed a so powerful theory, that he applied it to any complex system. He defined a canonical form of systems (see Fig. 1) as an interaction of three sub-systems : Decision System, Information System and Operating System.

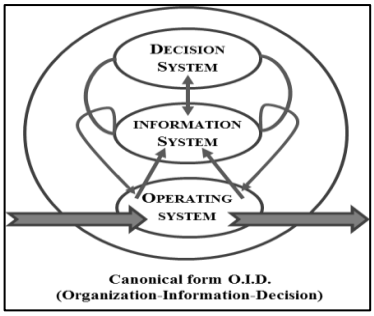

Fig. 1. Interaction between Decision, Information and Operating Systems. (reproduced from [28])

This point of view has been a guide for our work, both for supporting designers to model robots at a high abstraction level, and for architecting the software which will be generated and uploaded in the EV3 bricks. The Decision System plays a fundamental role in our project, since it represents the controller of the robot. So, a special attention has been carried on for its representation.

The Information system is a representation elaborated by the system of its environment, according to its mission. It can be viewed as a memory of its experience and its past events which have been received or sent. This memory, depending on the maturity of the system, may be seen as an experience or an intelligence that may have influence on the Decision System.

Lastly, the Operating System is the lower level system that copes with the environment and represents the operative part of the system allowing actions to be performed, events

to be received or sent, and system to act under the Decision System control.

C. *Behavioural Modelling using State Machines*

State machine formalism belongs to the standard UML. It is suitable for modelling the controller with respect to MDA approach for many reasons. First one is that this formalism is appropriate to model reactive systems that must be aware of their environment and have to permanently listen incoming events and react in a proper way.

State machine formalism allows system finality to be defined at high abstraction level. States represent the system steps that can be associated to activities during its mission, even when details are not yet designed. The transitions point out the evolution of the system triggered by external events (i.e. operator action through a remote interface, events coming from another robot, etc.) or internal ones (a red line has been detected by the color sensor of the robot, the robot has found an obtsacle). Transitions allow also to define events sent back by the controller to its environement (inside the robot or outside). Operating and Decision Systems communicate through the Information System that is in charge to manage and store the events exchanged between the robot and its environment.

Lastly, this formalism allows the abstraction refinement defined in MDA approach, since states may be again represented by sub state-machines. This hierarchical design mechanism is a way to apply refinement of models [29] that can be formally verified to maintain liveness and safety properties.

D. *Meta Modelling and Model Transformation*

These concepts are key words of MDA. Meta modelling is a way to define the domain concepts, their relations, and rules to manipulate them. It is thus a way to structure a project, and for our project, a support to define requirements of the framework RMDA. The meta model has 2 advantages: it allows a textual or graphical language to be defined with respect to pre-defined domain concepts ; it ensures that user's models will conform to meta model.

Furthermore, when the meta model itself conforms to a meta meta model (such as *ecore*, proposed by Eclipse Foundation), the generic modelling tools of Eclipse Foundation can be reused and applied to specific project, that saves effort and reduces errors.

Model transformation is also a benefit of the MDA approach: new models may be automatically generated from a given meta model to another by highlighting transformation rules between their concepts. For example, model transformation into JAVA code is a way to refine a design model into a low level model, i.e. an executable code suitable for a given OS. This mechanism again saves effort and reduces errors.

We explain in the next section how the concepts of MDA and systemic modelling have been applied to set up the framework MDA.

IV. Implementation of the Framework RMDA

The framework RMDA focuses on the design of a robot to which a mission has been assigned. The second goal is the generation of the Java code skeleton which, when completed, will be embedded in the EV3 brick. In this section, we will not consider the modelling of the environment that is interacting with the robot. However, its corresponding representation is defined in a same way than the robot by defining at a high abstraction level what kind of interactions may exist between them. The environement may be considered at a high abstraction as an interface with provided and required services. In this section, we present three parts that match with the main steps of our project that are: the meta modelling of the robot concepts, the textual and graphical design environment RMDA, and RMDA2JAVA, the code generator, which again, is based on model transformation.

A. *Meta model associated with LEGO Mindstorms Robots*

The first step that has been done concerns the definition of the meta model allowing any robot to be designed taking into account the aforementioned conceptual requirements: separation of concerns, abstraction and systemic points of view. We have thus define the robot as being the assembly of three components corresponding to the Decision System, the Information system and the Operating System (see Fig.2). Each of this component has its own ports representing the physical point of connexion of the robot sub-components. In order to appreciate the meta model, we first present the physical parts of LEGO Minstorms robot, available in the conventional package for ROBAFIS contest. It includes LEGO pieces allowing the robot to be built up. There are several kinds of sensors: color, touch, gyroscopic and ultrasonic, and motors of two kinds: large and medium. The EV3 brick has a ARM9 CPU with a LINUX system and 64 MB of memory.

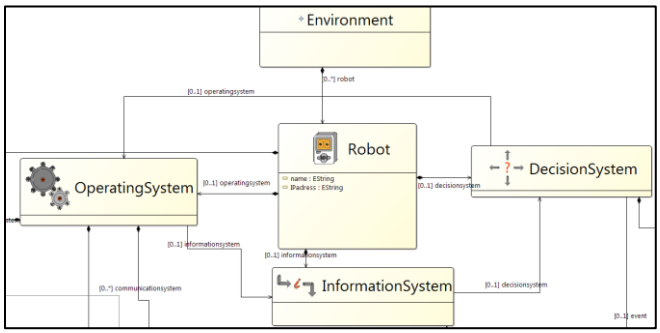

Fig. 2. Meta model of a Robot considered as an assembly of three abtract systems.

The Decision System is modeled by a state machine (see Fig. 3). For this first experiment, we do not take into account all features of state machines defined in UML standard [31]. They are restricted to states (including initial and final states), and transitions composed of a trigger and an action. We limit the definion of triggers and actions to event specifications. By this way, the decision system may analyze the incoming event and, depending on the state of the robot, the outcoming transition specifies what event has to be sent, and on which port.

The Operating System is constituted by some sub-systems that are usually embedded on robots such as a Moving System, a Grabbing System and a Communication System (Fig. 4). Moving and Grabbing systems include motors that are connected through ports to the Decision System. The communication system is embedded in the EV3 brick and its connexion is not visible by the designer. Several

communication systems are available: wifi, bluetooth or by wire. Services offered by the Operating System are modeled by a list of methods belonging to required or offered interfaces. For example, the moving system may offer services such as go right, go back, turn right, turn left. These services will have to be refined according to the number of motors chosen by the designer and their specification. The color sensor may use a required interface to send a signal about a given color recognition.

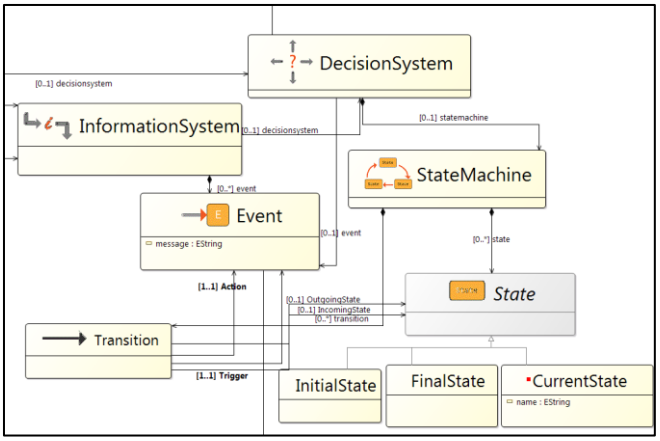

Fig. 3. Meta model of the Information and Decision Systems.

The Information System is the memory of the robot. It keeps trace of incoming events of the Decision System and outgoing events . The link is established in the meta model through the Event Manager that handles Events which are linked with Trigger and Action classes associated with Transitions of the State Machine class.

Fig. 5 points out in a hierarchical view all concepts of the meta model.

B. *RMDA, a Graphical Framework for Designing Robots*

Having defined the meta model, we have to set up the environment allowing the creation of models associated with the defined concepts. We made the choice to develop our framework in the Eclipse environment. By this way, we take benefit of tools such as EMF (Eclipse Modelling Framework) or GMF (Graphical Modellig Framework) [32]. The first one offers tools that support designers for creating models that will, by construction, conform to a given meta model. The interface allows a modelling class to be selected and instanciated. Its properties may be assigned by the user through a tabular textual interface. This is a first way to create models.

Another way, that is less abstract for the designer is to get a graphical interface. An interesting project named Sirius [10] have been developed by Obeo Designer Company allowing customized graphical editors to be generated. The generation requires the definition of a meta model and a description file setting up graphical parameters (icons for example), construction rules (for linking items for example) and actions associated with the handling of the classes of the meta model (verification for example that the system has no many decision systems).

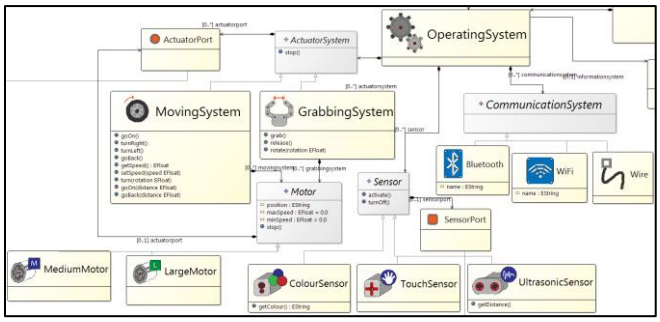

Fig. 4. Meta model of the Operating System.

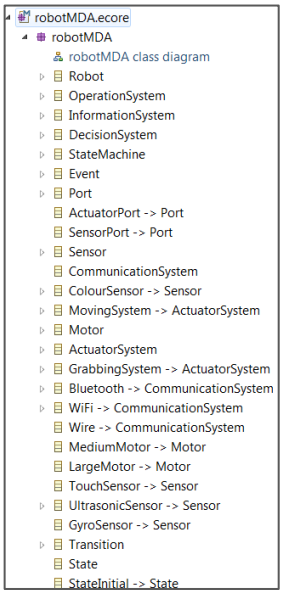

Fig. 5. Main classes of the meta model of LEGO Mindstorms.

This project has been useful for us in order to produce our framework RMDA. Fig. 6 gives an overview of the process of RMDA generation by Sirius and its use for creating a specific model named MyRobot.

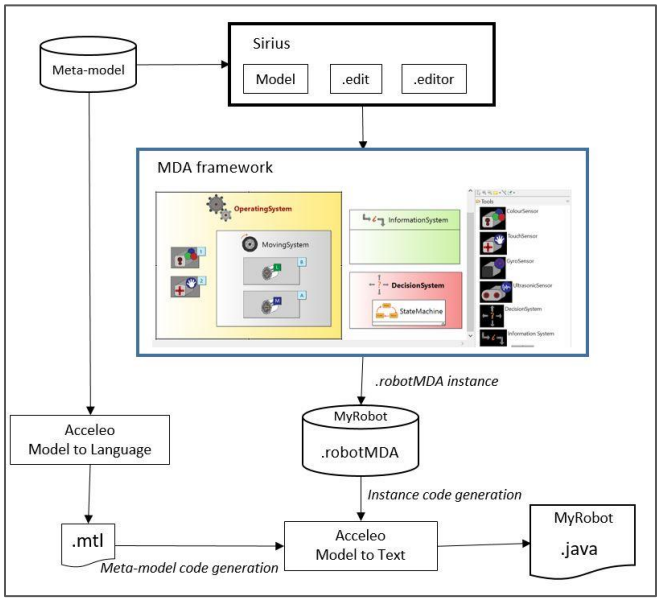

Fig. 6. Process for generating RMDA Framework and generating JAVA code.

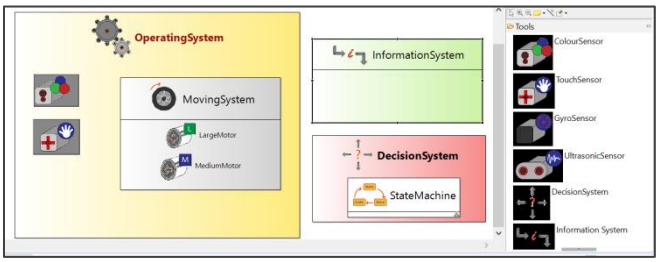

Fig. 7. RMDA Framework and MyRobot, a user model.

Fig. 7 gives an overview of the first release of the RMDA Framework. On the right part, one can see the control element palette window whose icons are matching the concepts of the meta model. For instance, at the top there are : Colour, Touch, Gyro and Ultrasonic Sensors icons. In the main window, MyRobot has been defined: it is constituted by three sub-systems, as defined in the conceptual requirements of section III B. The Decision System is associated with a state machine, the Operating System is constituted by two Sensors and a Moving System which has two Motors, that is conform to the meta model represented in Fig. 4.

C. *RMDA2JAVA, the Generator of JAVA code to control LEGO Mindstorm Robots*

This last section is the final step of our project, since its goal is to generate the JAVA code that will be embedded in the controler of the robot. Again, this generation is based on a model transformation supported by the Acceleo tool [33]. The generation is performed by a set of rules expressed in the Acceleo MTL (Model to Text Language) [34] .

There are two steps for generating the final embedded code:

1. Getting the JAVA code of the meta model, that corresponds to the pre-defined classes of the project. This part is thus done once, when the meta model is stable.
2. Getting the JAVA code of the robot for a specific mission. This part is thus specific to any designed model.

Fig. 8 shows a part of the Acceleo MTL code allowing the meta model concepts in relation with motor LEGO Minstorms concepts to be transformed into JAVA, taking into account the LeJOS library. As pointed out in Fig. 8, motors are of two types: MediumMotor or LargeMotor. Let us precise that for this kind of items, the port is unique and is automatically created in the generated code.

Fig. 9 points out the JAVA classes that have been generated and the architecture of the project that conforms to the systemic decomposition of the system in three sub-systems: decision, information and operation.

The same kind of rules have been defined in order to generate the JAVA skeleton of the controler. An interesting feature of this code is that it offers to the team of design and development a stable well organized architecture with pre-defined packages that constitute a guide to complete the code. Fig. 10 illustrates the JAVA code automatically generated from the description of MyRobot.

```
[comment]genAttrClass: template of the body "public class
" + class.name with the generation of its attributes
[/comment]
[genAttrClass(aMotor)/]
[comment]Generation of the specific attributes for
motors, within the Lejos environment[/comment]
[if (aMotor.name='LargeMotor')]
        EV3LargeRegulatedMotor motor;
[elseif (aMotor.name = 'MediumMotor')]
          EV3MediumRegulatedMotor motor;
[/if]
[comment] Generation of the constructor of the two
subclasses of Motor: MediumMotor and LargeMotor
[/comment]
[if(aMotor.name='MediumMotor' or
       aMotor.name='LargeMotor')]
public [aMotor.name/](ActuatorPort port){
[if (aMotor.name = 'LargeMotor')]
        this.motor = new EV3LargeRegulatedMotor(port);
[elseif (aMotor.name = 'MediumMotor')]
        this.motor = new EV3MediumRegulatedMotor(port);
[/if]
}
[/if]
[comment] Generation of setters and getters for each
attribute of the class[/comment]
[genSetGet(aMotor)/]
```

Fig. 8. Accelo MTL code for generating JAVA classes corresponding to RobotMDA meta model (extract)

At the present time, R&D mission is not yet achieved. The transformation of state machine has not yet been implemented. However, the pattern of code is defined, since the first task performed by students has been to learn how to program a controler in LeJOS environment when it is modeled by a state machine. They have been trained to thread programming with event handling, and are thus prepared for code generation.

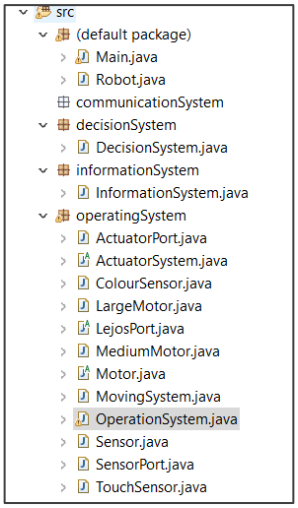

Fig. 9. Architecture of the JAVA classes corresponding to the meta model.

V. Conclusion: Feedback and Perspectives

In this paper, we have described in detail the steps of an experiment, from conceptual to practical points of view, for training students at the beginning of their Software Engineering curriculum. The goal was to learn the model driven engineering approach and how applying its concepts in the context of programming LEGO Mindstorms robots.

```
import decisionSystem.DecisionSystem;
import informationSystem.InformationSystem;
import lejos.hardware.lcd.LCD;
import lejos.utility.Delay;
import operatingSystem.*;
import lejos.hardware.port.*;

public class Main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        LCD.drawString("Plugin test",0,4);
        Delay.msDelay(5000);
        MediumMotor rightmotor = new MediumMotor(new ActuatorPort(MotorPort.A));
        LargeMotor leftmotor = new LargeMotor(new ActuatorPort(MotorPort.B));
        MovingSystem MS = new MovingSystem (rightmotor, leftmotor, 2);
        InformationSystem IS = new InformationSystem();
        DecisionSystem DS = new DecisionSystem();
        OperationSystem OS = new OperationSystem();
        OS.setActuatorsystem(MS);
        OS.setInformationsystem(IS);
        DS.setOperationsystem(OS);
        IS.setDecisionsystem(DS);
        Robot MyRobot = new Robot(IS,DS,OS);

    }

}
```

Fig. 10. The main function automatically generated and corresponding to MyRobot description.

Even if the project is yet under development, this experiment is a success from a practical point of view: the RMDA Framework allows user-oriented models to be defined. Some enhancements are planned for the end of this month to complete the graphical modelling of the state machine associated with the controler and complete the JAVA generation of the state machine. Let us note that it is also possible to reuse in the RMDA Framework existing environments (such as Obeo Designer). This option has not been chosen for this first step in order to make students responsible of the entire realization. In the future, we could plan to have an advanced release of the tool and connect it with Obeo Designer. The project was also beneficial to students for learning how programming reactive system using JAVA threads. This task has been their first practical training with their LEGO Mindstorms robot. It was controlled using a remote graphical interface and was autonomous to stop when obstacles were detected.

Another conclusion of this experiment is that the project has been a great experience for students to learn, understand and apply abstract concepts of modelling and meta modelling that are usually difficult to handle and require times and experience. They have been trained to take references in standard specifications such as EMF, GMF and UML. They have learned how to set up a bibliography and write a paper.

Lastly, this experiment has also been a great experience for the instructor of the project. Students have been invested in the project and open to discussion. RMDA Framework can be again improved. Nevertheless, it will be useful for students of the System Engineering Department that focus on the physical architecture design of the robots and behaviour modelling.

The advantages of MDA versus conventionnal software development have been often discussed according to two points of view: reducing the cost of development and increasing the quality of software [35]. The entire software community does not agree about these benefits. Nevertheless, some studies point out that model driven development yields better quality for softwares having complex problems to be solved (from functional point of view) [36]. From our point of view, what has to be expected from MDA is to support designers for both modelling and verifying models before their implementation. Modelling requires to be aware of the precise semantics of modelling languages, that is lacking at the present time. Models have not to be considered as sketches of a system but as formal specifications allowing verifications and code generation.

Future R&D missions will consist at first in completing the generated software to be uploaded on the robot. That is achieved by increasing the level of modelling details, specially in the state machine associated with the Decision System. This part must be also completed by testing the robot behaviour in several situations. The second goal will be to train students in the field of formal verification, which does not usually attract them. The goal is to set up a bridge between the RMDA Framework and our current research on conformance relation verification including liveness and safety analyses [25, 29]. That means it is better to be sure the model is correct according its specification before generating the code.

REFERENCES

[1] LEGO, "LegoMindStorm EV3," 2019. [Online]. Available: https://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313.

[2] AFIS, "Association Française de l'Ingénierie Système," 2019. [Online]. Available: https://www.afis.fr/pages/accueil.aspx. [Accessed: 09-May-2019].

[3] INCOSE, "International Council on Systems Engineering Website," 2019. [Online]. Available: https://www.incose.org/. [Accessed: 09-May-2019].

[4] AFIS, "L'Ingénierie Système appliquée à la réalisation d'un robot," 2019. [Online]. Available: http://www.robafis.fr/RobAFIS/ Bienvenue.html. [Accessed: 09-May-2019].

[5] T. Griffin, The art of LEGO® Mindstorms EV3 programming, No Starch. No Starch Press, 2014.

[6] F. Truyen, "The fast guide to Model Driven Architecture," White Paper, Cephas Consulting Corp., 2006.

[7] E. A. Gandy, S. Bradley, D. Arnold-Brookes, and N. R. Allen, "The use of LEGO Mindstorms NXT Robots in the Teaching of Introductory Java Programming to Undergraduate Students," Innov. Teach. Learn. Inf. Comput. Sci., vol. 9, no. 1, pp. 2–9, Feb. 2010.

[8] P. J. Bradley, J. A. De La Puente, J. Zamorano, and D. Brosnan, A platform for real-time control education with LEGO MINDSTORMS®, vol. 9, no. Part 1. IFAC World Congress, 2012.

[9] J. Dupont and F. Madiot, "Mindstorms Robot Tutorial." [Online]. Available: https://wiki.eclipse.org/Sirius/Tutorials/Mindstorms. [Accessed: 09-May-2019].

[10] Obeo Designer, "Eclipse Sirius - Obeo Designer." [Online]. Available: https://www.obeodesigner.com/en/product/sirius. [Accessed: 09-May-2019].

[11] R. Passama, D. Andreu, D. Crestani, and K. Godary-dejean, "Architectures de contrôle pour la robotique - Approches et tendances," Tech. l'Ingénieur, vol. 33, no. 0, 2014.

[12] P. Trojanek, "Model-driven engineering approach to Design and implementation of robot control system," in International Workshop on Domain-Specific Languages and models for ROBotic systems, 2011.

[13] L. Fabresse, N. Bouraqadi, C. Dony, and M. Huchard, "A language to bridge the gap between component-based design and implementation," in Computer Languages, Systems and Structures, vol. 38, no. 1, Elsevier, 2012, pp. 29–43.

[14] D. Thomas, C. Baron, and B. Tondu, "Ingénierie dirigée par les modèles appliquée à la conception d'un contrôleur de robot de service," Idm06, 2006.

[15] C. Schlegel, A. Steck, and A. Lotz, Model-Driven Software Development in Robotics: Communication Patterns as Key for a Robotics Component Model. iConcept Press, 2011.

[16] Oracle, “Lesson : Concurrency.” [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/concurrency/.

[17] E. Estévez, A. Sánchez-García, J. Gámez-García, J. Gómez-Ortega, and S. Satorres-Martínez, A novel model-driven approach to support development cycle of robotic systems, vol. 82, no. 1–4. IFAC, 2016.

[18] H. Baumeister, F. Hacklinger, R. Hennicker, A. Knapp, and M. Wirsing, “A Component Model for Architectural Programming,” Electron. Notes Theor. Comput. Sci., vol. 160, no. 1, pp. 75–96, 2006.

[19] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane, “RobotML, a Domain-Specific Language to Design , Simulate and Deploy Robotic Applications,” in SIMPAR’12 Proceedings of the Third international conference on Simulation, Modeling, and Programming for Autonomous Robots, 2012, pp. 149–160.

[20] C. Pons, G. Pérez, R. Giandini, and G. Baum, “Applying MDA and OMG Robotic Specification for Developing Robotic Systems,” LNCS, vol. 9959, pp. 51–67, 2016.

[21] X. Blanc, J. Delatour, and T. Ziadi, “Benefits of the MDE approach for the development of embedded and robotic systems Application to Aibo,” Proc. 2nd Natl. Work. “Control Archit. Robot. from Model. to Exec. Distrib. Control Archit. CAR. 2007, no. 2, 2007.

[22] M. Alessandro and A. Le, “Etude de patterns de systèmes réactifs à l’aide d’un LEGO Mindstorms: Dossier de réalisation ,” IMT Mines Ales, 2017.

[23] F. Matheus Fernandes de Oliveira, “Report Case Study: State Machine’s implementation methodology using Java and LeJos Ev3,” IMT Mines Ales, 2018.

[24] LeJOS, “LeJOS, Java for Lego Mindstorms / EV3.” [Online]. Available: http://www.lejos.org/ev3.php. [Accessed: 14-May-2019].

[25] T. Lambolais and A.-L. Courbis, “Development and Verification of UML Architectures by Refinement and Extension Techniques,” in European Congress on Embedded Real Time Software and Systems (ERTS2), 2018.

[26] O. Vogel, I. Arnold, A. Chughtai, and T. Kehrer, “Architecture Means,” in Software Architecture: a Comprehensive Framework and Guide for Practitioners, Springer S., 2011.

[27] OMG, “OMG Document -- ormsc/14-06-01 (MDA Guide revision 2.0),” 2014. [Online]. Available: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01. [Accessed: 15-May-2019].

[28] J.-L. Le Moigne, “Formalism of systemic modelling,” in Some physicochemical and mathematical tools for understanding of living systems, R. Greppin H., Bonzon , M., Degli Agosti, Ed. University of Genova, 1993, pp. 367–368.

[29] T. Lambolais, A.-L. Courbis, H.-V. Luong, and C. Percebois, “IDF: A framework for the incremental development and conformance verification of UML active primitive components,” J. Syst. Softw., vol. 113, pp. 275–295, Mar. 2016.

[30] D. Eriksson, “A principal exposition of Jean-Louis Le Moigne’s systemic theory,” Cybern. Hum. Knowing, vol. 4, no. 2–3, pp. 33–77, 1997.

[31] OMG, Unified Modeling Language Superstructure 2.2. 2007.

[32] The Eclipse Foundation, “Graphical Modeling Framework (GMF).” [Online]. Available: https://www.eclipse.org/modeling/gmp/. [Accessed: 15-May-2019].

[33] Eclipse Foundation, “Eclipse Acceleo.” [Online]. Available: https://projects.eclipse.org/projects/modeling.m2t.acceleo. [Accessed: 15-May-2019].

[34] Eclipse Foundation, “Eclipse Modeling - M2T.” [Online]. Available: https://www.eclipse.org/modeling/m2t/?project=acceleo. [Accessed: 15-May-2019].

[35] R. Picek and V. Strahonja, “Model Driven Development - future or failure of software development,” 18th Int. Conf. Inf. Intell. Syst., 2007.

[36] J. I. Panach Navarrete, O. Dieste, B. Marin, S. España, S. Vegas, O. Pastor, N. Juristo, “Evaluating Model-Driven Development Claims with respect to Quality: A Family of Experiments,” IEEE Trans. Softw. Eng., December 2018.