Applying Software Engineering Techniques to the Development of Robotic Systems

Claudia Pons[1,2], Gabriela Arévalo[1,2], Gonzalo Zabala[1], Ricardo Morán[1]

[1] CAETI - UAI,
Buenos Aires, Argentina
[2] CONICET
Avda. Rivadavia 1917
(1033), Buenos Aires, Argentina
{gabrielag.arevalo, claudia.pons,gonzalo.zabala}@uai.edu.ar,
richi.moran@gmail.com

Abstract. In these days most robotic systems tend to be complex to maintain and reuse because existing frameworks are based mainly on code-driven approaches. This means the software development process is reduced to the implementation of systems using specific programming languages. During the constant evolution, the systems grow in size and in complexity. Even when these approaches address the needs of robotic-focused markets, currently used methodologies and toolsets fail to cope with the needs of such complex software development process. The general objective of our project is the definition of a methodological framework supported by a set of tools to deal with the requirements of the robotic software development process. A major challenge is to make the step from code-driven to model-driven in the development of robotic software systems. Separating robotics knowledge from short-cycled implementation technologies is essential to foster reuse and maintenance. In this paper we report our initial results.
Keywords: robotic software system, software development process, software engineering.

1 Introduction

Robotic systems (RSs) play an increasing role in everyday life. The need for robotic systems in industrial and educational settings increases and becomes more demanding. While robotic systems grow to be more and more complex, the need to apply the engineering principles to their software development process is a major challenge in these days. Traditional approaches, based on mainly coding the applications without using modelling techniques, are used in the development process of these software systems. Even when the applications are running and being used in the different robotic systems, we identify several problems. Among them, it is worth mentioning there is no clear documentation of design decisions taken during the coding phase, making the evolution and the maintenance of the systems difficult. When using specific programming languages, such Smalltalk in

EToys [3], or C in RobotC [4], we lose the possibility of generalizing concepts that could be extracted, reused and applied in different systems, avoiding to code them from scratch when they are needed.

Thus, we observe that currently used methodologies and toolsets fail to address the needs of such complex software development process. It is widely accepted that new approaches should be established to meet the needs of the development process of actual complex RSs. Component-based development (CBD) [20], Service Oriented Architecture (SOA) [8] [9], as well as Model Driven software Engineering (MDE) [19] [15] and Domain-Specific Modeling (DSM) [13] are the main modelling and composition-based technologies in the RSs domain. In our project, we will investigate on the current use of those modern software engineering techniques to improve the development of robotic software systems and their actual automation level. Considering that existing systems are already coded, a major challenge is to make the step from code-driven to model-driven in the development of robotic software systems to extract the general and specific concepts of existing applications based on the different specific programming languages.

The general objective of our research and development project is the definition of a methodological framework (composed of models and code) supported by a set of tools able to deal with the requirements of the robotic software development process and considering the existing implemented approaches. Robotic platforms must possess a highly dynamic adaptive capacity, accompanying the rate of development of such technologies and the specific features of each hardware platform.

2 Specific Objectives and Working Hypothesis

In this project we are working on the following hypothesis:

- It is mandatory to work towards applying engineering principles to cope with the complexity of existing implemented robotic software systems because actual systems are more focused on hand-crafted single-unit systems.
- Interfaces and behavior of the robotic systems should be defined at a higher level of abstraction so that they could be re-used with different platforms. Separating robotics knowledge from short-cycled implementation technologies is essential to foster reuse and maintenance.
- Applying existing software engineering modelling methodologies, such as MDE, SOA and CBD, to build robotic software systems will save a great amount of time and effort while favouring reusability and maintenance of such systems.

Within this context, the specific objectives of our project are:

[3] http://www.etoys.com/
[4] http://www.robotc.net/

– Summarizing the existing state of the art concerning the application of software engineering modelling methodologies, such as SOA, MDE and CBD on the robotic systems development field;
– Building a methodological approach on top of the applications of existing techniques providing an advance in the field;
– Building tool support to the robotic software development process. Examples of these tools are: a domain specific modeling language equipped with graphical editors, code generation facilities, integration with web services and component definition editors.
– Addressing the results to build real robotic systems used in industry and education.

3 Problem Relevance: Existing Approaches

Although the complexity of robotic software is high because it is mainly based on specific programming languages, reuse is still restricted to the level of libraries. At the lowest level, different libraries have been implemented for robotic systems to perform tasks, such mathematical computations for kinematics, dynamics and machine vision [10]. Instead of composing systems out of building blocks with proved services, the overall software integration process for another robotic system often is still reimplementation of the glue logic to bring together the various libraries. Often, the overall integration is completely driven by middleware systems and their functionalities. Middlewares are often used to hide complexity regarding inter-component communication, such as OpenRTM-aist [4]. Obviously, this approach is expensive and does not take advantage from a mature process to enhance overall robustness. We have faced with this problem in our own practice. We have been programming educational robots for more than 10 years [3] [2] and we have observed in the last years the emergence of robotic kits oriented to non-expert users gave rise to the development of a significant number of educational projects using robots. Those projects apply robots at different education levels, from kindergarten through higher education, especially in areas of physics and technology. In this context, one of the problems we have found is that the hardware of the robotic kits is constantly changing; in addition its use is not uniform across different regions and even education levels. Therefore, the technical interfaces of these robots should hide these differences so that teachers are not required to change their educational material every time they are used. An example of these interfaces is "Physical Etoys" [2], a project which we participated in and which proposes a standard teaching platform for programming robots, regardless of whether they are based on Arduino, Lego, or other technologies. In this context, it is widely accepted that new approaches should be established to meet the needs of the development process of today complex RSs. Component-based development (CBD) [20], Service Oriented Architecture (SOA) [8] [9], as well as Model Driven software Engineering (MDE) [15] and Domain-Specific Modeling (DSM) [13] are the most relevant technologies in the RSs domain. Firstly, the Component-based development paradigm [20] states

that application development should be achieved by linking independent parts, the components. Strict component interfaces based on predefined interaction patterns separates the functionalities in different parts, and thus partition the overall complexity. This results in loosely coupled components that interact via services with contracts. Components, such as architectural units, allow specifying very precisely, using the concept of port, both the provided services and the required services by a given component and defining a composition theory based on the notion of a connector. Component technology offer high rates of reusability and ease of use, but little flexibility regarding to the implementation platform: most existing component are linked to C/ C++ and Linux (e.g. Microsoft robotics developer studio [1], EasyLab [7]), although some achieve more independence, thanks to the use of some middleware (e.g. Smart Software Component model [16], Orocos [10]). Secondly, we need to define interfaces and behavior at a higher level of abstraction so that they could be used in systems with different platforms. This is what prompted the idea of abstract components, which would be independent of the implementation platform but could be translated into an executable software or hardware component. Thus, the migration from code-driven designs to a model-driven development is mandatory in robotic components to overcome the current problems. A model-based description is a suitable means to express contracts at component interfaces and to apply tools to verify the overall behavior of composed systems and to automatically derive the executable software. Instead of building tool support for each framework from scratch, one should now try to either express the needed models in standardized modeling languages like UML or any DSL, separating components from the underlying computer hardware. In the context of software engineering, the Model Driven Development (MDD) [19] [15] and Domain-Specific Modeling approach (DSM) [13] have emerged as a paradigm shift from code-centric software development to model-based development. Models are considered as first-class constructs in software development, and developers' knowledge is encapsulated by means of model transformations. The essential characteristic of MDD and DSM is that software development primary focus and work products are models. Its major advantage is that models can be expressed at different levels of abstraction and hence they are less bound to any underlying supporting technology. Finally, Service-oriented architecture (SOA) [8][9] is a flexible set of design principles used during the phases of systems development and integration in computing. A system based on a SOA will package functionality as a suite of interoperable services that can be used within multiple, separate systems from several business domains. SOA also generally provides a way for consumers of services, such as web-based applications, to be aware of available SOA-based services. SOA defines how to integrate widely disparate applications for a Web-based environment and uses multiple implementation platforms. Rather than defining an API, SOA defines the interface in terms of protocols and functionality. Service-orientation requires loose coupling of services with operating systems, and other technologies that underlie applications. So far, there is no proposal taking advantage of the combined application of CBP, SOA and MDE neither

to robotic software system development in general, nor to educational robotic system development in particular.

4 First results: Modeling and automatic code derivation

The MDD approach represents a paradigm where models of the system, at different levels of abstraction, are used to guide the entire development process. Models are implementation-independent and they are automatically transformed to executable code. The MDD process can be divided into three phases: the first phase builds a platform independent model (PIM), which is a high-level technology-independent model; then, the previous model is transformed into one or more platform specific models (PSM); these models are lower level and describes the system in accordance with a given deployment technology; finally, the source code is generated from each PSM. As said in section 1, most systems are coded without documentation or designed models. In this section we show how we could have MDD process for automatically deriving from the existing code of an already implemented robotic system with a reverse-engineered approach.

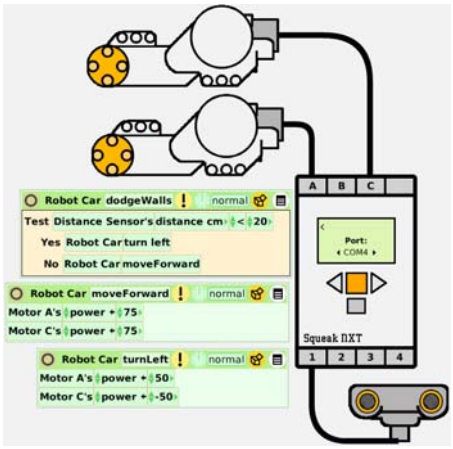

Fig. 1. Implementation of the Robot with Etoys in a visual way

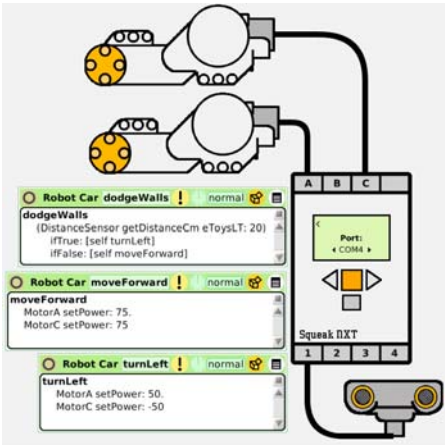

Fig. 2. Implementation of the Robot with Etoys using code

To illustrate our approach, we use a small example of a 4-wheel robot, which is composed of a distance sensor and two motors A and B. The robot moves straightforward constantly, while there are not obstacles on its way. Whenever the robot finds an obstacle, it turns left to avoid the obstacle and keeps moving. To find the obstacle, the distance sensor will detect if the robot has a wall in a distance less than 20 cm. If so, the robot will change the power of the motors to make them turn left. If there is no wall, the motor keeps with the same value.

Depending on the existing platforms, there are different ways we implement this robot behavior. We will show Physical Etoys[5], RobotC[6] and Pharo [7].

Physical Etoys is a visual programming tool that connects the virtual world of computers with the real world in which we live in. With Physical Etoys you can program real world objects (such as robots) to perform interesting tasks, or you can sense the world and use that information to control virtual objects (such as drawings on the screen). The user must grab tiles representing instructions and assembling a script. Figure 1 shows the visual representations of our example using Physical Etoys. If you do not use the predefined tiles to build the script, you can code explcitly the robot and its behavior using Smalltalk[8](the embedded programming language) as shown in Figure 2. Robotc is an Integrated development environment targeted towards students that is used to program and control LEGO NXT's, VEX's, and RCX robots using a programming language based on C. It aims to allow code to be ported from one robotics platform to another with little or no change in code. You do not have a visual programming environment in RobotC, all robot behaviour must be defined by coding in C as shown in Figure 3. If you use an embedded robot framework in existing programming languages, in our case Pharo (a free open-source Smalltalk environment), you can also code the robot behaviour. Figure 4 shows how we code the example using Pharo. Depending on the abstraction level of the programming languages, sometimes we need to deal with specific details of the implementation. For example, in Pharo we code explicitly how to connect the port and plug the motors previously to specify the desired behavior. These connections are implicit in other platforms.

If we need to represent our example in another platform, we must provide some code transformation from one platform to another one, or even build the application from scratch. But this process is expensive. Our proposal is to build a PIM that allows to abstract the domain concepts and their functionalities using MDD and CBSD. With the generated models we can then derive the code in any specific robotic language. Thus, in our example, we can identify the components Robot, DistanceSensor and Motor, and the functionality is as we described previously.

We can represent this robot with a Component and Behaviour Models represented with their respective UML models. Figure 6 shows a UML Component Diagram that identifies the structural components of the example and which are the required/provided interfaces in their connectors, and Figure 5 shows an Activity Diagram to model the behaviour of the robot example. Even though these models are useful enough to understand the existing implementation and show the transformation of PSM (code) to PIM (Component and Behavior Models), we could have an intermediate PSM model of objects (due to the fact we are working with object-oriented code) represented with a Class Diagram inferring

[5] http://tecnodacta.com.ar/gira/projects/physical-etoys/
[6] http://www.robotc.net
[7] http://www.pharo-project.org
[8] www.cincomsmalltalk.com

```
task main()
{
    SetSensorLowspeed(IN_1);
    Wait(500);
    while(true)
    {
        if(SensorUS(IN_1) < 20)
        {
            girar();
        }
        else
        {
            avanzar();
        }
    }
}

safecall void avanzar()
{
    OnFwd(OUT_A, 75);
    OnFwd(OUT_C, 75);
}

safecall void girar()
{
    OnFwd(OUT_A, 50);
    OnFwd(OUT_C, -50);
}
```

Fig. 3. Implementation of the Robot with Nxc

```
| nxt motorA motorC sensorDeDistancia |
nxt := LegoNxt new connectOnPort: 'COM4'.

motorA := NxtMotor new plugOn: nxt portA.
motorC := NxtMotor new plugOn: nxt portC.
distanceSensor := UltrasonicSensor new plugOn: nxt port1.

[distanceSensor rawValue < 20
    ifTrue: [motorA power: 50.
        motorC power: -50]
    ifFalse: [motorA power: 75.
        motorC power: 75]] repeat.
```

Fig. 4. Implementation of the Robot with Pharo

the classes and their relationships based mainly on the code. This last step can be a semi-automatic approach using the initial work done in Passerini work [14]. Due to the space limitations of the paper, we will not present the corresponding Class Diagram of our example. Thus, with an abstraction of the concepts represented mainly in Component and Behavior models, we can generate semi-automatically the example in another robot programming language considering that we can have glue code to fill in the code (as we have shown in Pharo).

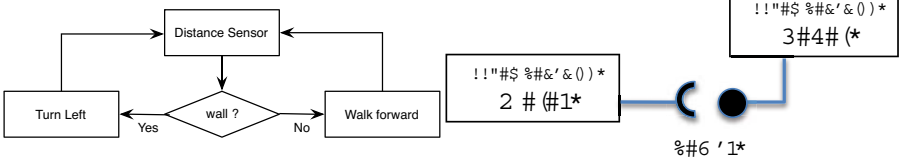

Fig. 5. Activity Model of the Behaviour of our Example

Fig. 6. Component Model of our Example

All the previous examples are built based on already implemented applications where the developer is able to access and modify -if needed- the code. However, the robot can be connected to external devices using predefined interfaces defined in those external components and whose implementation the developer can not access in the environment he/she is working on. In our exam-

ple, Figure 8 (Pharo code) shows how we code and connect our example robot to an external component named `Kinect Server` that will indicate if it moves forwards or backwards depending the movements of arms of the user. In this specific case, we are not able to see the code of the component that models those arms' movements. However, in Pharo we can design the interfaces that will connect to the external component. In this specific case, the code is `kinect := KinectServer new connect`. However, in Physical Etoys we can design the interfaces that will connect to the external component using a visual representation and code (Figures 9 and 10). In our case, they are blue points represent the connections to the external device that deals with arms movements. When we design the interfaces we mean we are not implementing the functionality of the external components, but more than we implement the glue code to be able to connect both components. Even though the internal and external identified components have a similar structure in the models, the way they connect to the robot are different because in the first case the developer implements their interfaces, and in the second case, the interfaces are already defined and the developer should be able to connect to them by implementing the corresponding glue code. In our specific case in Physical Etoys, it is represented with blue points.

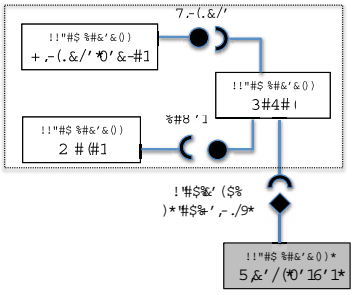

Fig. 7. Component and Service Models with External Interaction

```
nxt := LegoNxt new connectOnPort: 'COM4'.
motorA := NxtMotor new plugOn: nxt portA.
motorC := NxtMotor new plugOn: nxt portC.

distanceSensor := UltrasonicSensor new plugOn: nxt port1.

kinect := KinectServer new connect.

kinect when: #skeletonUpdate evaluate: [:skeleton |
        motorA power: skeleton rightHand y - skeleton rightShoulder y.
        motorC power: skeleton leftHand y - skeleton rightShoulder y].

[distanceSensor rawValue < 20
        ifTrue: [nxt playSoundFile: 'Woops.rso']] repeat.
```

Fig. 8. Implementation of the Robot with Pharo

It is then worth to be able to identify two models: the Component Model shows the internal components, and we build a Service Model that shows the external components. Thus, Figure 7 shows the Component and the Service Models together. In our specific case, our service model is reduced to only one component. In more complex platforms, we can have several services that can be modelled with their respective glue code to be connected to the implemented robots.

Summarizing, based on existing implementations we propose to infer the structure and behavior of the robots into class, activity and component/service models. These models are PIMs that can generate then new implementations

of the robots in another specific programming language, keeping the abstract concepts in these models and specific features in PIMs (such as code).

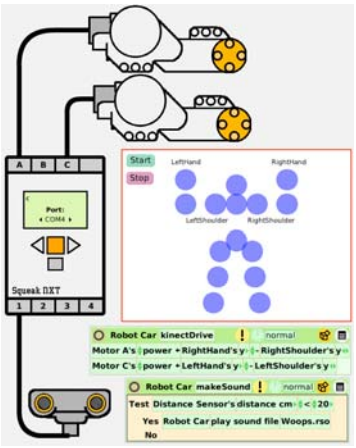

Fig. 9. Implementation of the Robot with Etoys in a visual way

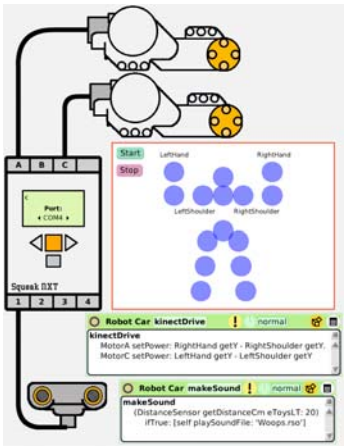

Fig. 10. Implementation of the Robot with Etoys using code

5 Conclusions and Future Work

In our project we are focused on capturing the fundamental concepts of the robotic software development process, its relationships and properties. This modeling approach includes concepts to represent services and components as primary elements in the robotic system in a higher level abstraction than the code itself.. So, the CBSD and SOA paradigms provide a starting point for a MDE approach in robotics where the differences between various software platforms and middleware systems can be completely hidden from the user due to the definition of intermediate abstraction level. The original contribution of this project consists in the development of a methodological framework supported with different tools for the construction of robotic software systems using mainly MDD. There are only preliminary proposal on applying model-driven development to robotics, see for example the works described in [17], [12], [7] [18]and [5]. None of these works takes advantage of the combination of the model-driven paradigm with service-oriented and component-based approaches, as we propose in this project. It is worth mentioning the Microsoft Robotics Studio (MSRS) [1], that is a service-oriented development and simulation platform to create robotics applications. MSRS is strongly based on the concept of SOA.

References

1. Microsoft: Microsoft robotics developer studio (March 2009), http://msdn.microsoft.com/en-us/robotics/default.aspx
2. Centro de Altos Estudios en Tecnología Informática (CAETI): Proyectos del Area Robótica (June 2011), http://www.caeti.uai.edu.ar
3. Gira Grupo de Investigación en Robótica Autónoma del CAETI: Physical Etoys (May 2011), http://tecnodacta.com.ar/gira/
4. Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., Yoon, W.: RT-middleware: Distributed component middleware for RT (robot technology). In: Proceedings of the International Conference on Intelligent Robots and Systems 2005 (IROS 2005). pp. 3933–3938 (2005)
5. Arney, D., Fischmeister, S., Lee, I., Takashima, Y., Yim, M.: Model-Based Programming of Modular Robots. In: Proceedings of 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC). pp. 66–74 (2010)
6. Barner, S., Geisinger, M., Buckl, C., Knoll, A.: EasyLab: Model-based development of software for mechatronic systems. In: Proceedings of the IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications. Beijing, China. IEEE (2008)
7. Bell, M.: Introduction to Service-Oriented Modeling. Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley and Sons (2008)
8. Bell, M.: SOA Modeling Patterns for Service-Oriented Discovery and Analysis. Wiley and Sons (2010)
9. Bruyninckx, H.: Open Robot Control Software: the OROCOS project. In: Proceedings of the 2001 IEEE International Conference on Robotics and Automation, ICRA 2001, Seoul, Korea. pp. 2523–2528. IEEE (2001)
10. Iborra, A., Caceres, D., Ortiz, F., Franco, J., Palma, P., Alvarez, B.: Design of Service Robots. Experiences Using Software Engineering pp. 24–33 (march 2009)
11. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling. John Wiley and Sons, Inc (2008)
12. Passerini, N.: Migration from inheritance to composition paradigms. In: Proceedings of the CACIC 2011 - ASSE Workshop (2001), submitted for evaluation
13. Pons, C., Giandini, R., Pérez, G.: Desarrollo de Software Dirigido por Modelos. Teorías, Metodologías y Herramientas. McGraw-Hill Education (2010)
14. Schlegel, C.: Communication patterns as key towards component interoperability. In: Software Engineering for Experimental Robotics (Series STAR, vol. 30), D. Brugali, Ed. Berlin, Heidelberg. pp. 183–210. Springer-Verlag (2007)
15. Schlegel, C., Hassler, T., Lotz, A., Steck, A.: Robotic Software Systems: From Code-Driven to Model-Driven Designs. In: Proceedings of ICAR 2009 International Conference on Advanced Robotics. IEEE Press (2009)
16. Son, H.S., Kim, W.Y., Kim, R.: Semi-automatic Software Development Based on MDD for Heterogeneous Multi-joint Robots. In: Proceedings of Future Generation Communication and Networking Symposia, 2008. FGCNS '08. pp. 93–98 (2008)
17. Stahl, M.V.: Model Driven Software Development. John Wiley and Sons, Inc (2006)
18. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edn. (2002)