Applying MDA and OMG Robotic Specification for Developing Robotic Systems

Claudia Pons[1,2,3(✉)], Gabriela Pérez[1], Roxana Giandini[1], and Gabriel Baum[1]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata, La Plata, Argentina
{cpons,gperez,giandini,gbaum}@info.unlp.edu.ar
[2] CIC, Comisión de Investigaciones Científicas PBA, La Plata, Argentina
[3] UAI, Universidad Abierta Interamericana, Buenos Aires, Argentina

Abstract. Robotics systems have special needs often related with their real-time nature and environmental properties. Often, control and communication paths within the system are tightly coupled to the actual physical configuration of the robot. As a consequence, these robots can only be assembled, configured, and programmed by robot experts. Traditional approaches, based on mainly writing the code without using software engineering techniques, are still used in the development process of these systems. Even when these robotic systems are successfully used, several problems can be identified and it is widely accepted that new approaches should be explored. The contribution of this research consists in delineating guidelines for the construction of robotic software systems, taking advantage of the application of the OMG standard robotic specifications which adhere to the model-driven approach MDA. Thereby the expert knowledge is captured in standard abstract models that can then be reused by other less experienced developers. In addition part of the code is automatically generated, reducing costs and improving quality.

Keywords: Robotic software system · Model driven software development · OMG standard

1 Introduction

Robotics systems are essentially real-time, distributed embedded systems. They have special needs often related with their real-time nature and environmental properties; they have to be able to cope with the uncertain and dynamic physical environment where they are immersed. Furthermore, robotic systems consist of different hardware components. There are a wide variety of controllers, sensors and actuators which results in very complex and highly variable architectures. Often, control and communication paths within the system are tightly coupled to the actual physical configuration of the robot. As a consequence, these robots can only be assembled, configured, and programmed by experts.

Traditional approaches, based on mainly coding the applications without using modeling techniques, are still used in the development process of these software systems. Even when the applications are running and being used in the different robotic systems,

J. Grabowski and S. Herbold (Eds.): SAM 2016, LNCS 9959, pp. 51–67, 2016.
DOI: 10.1007/978-3-319-46613-2_4

several problems can be identified. On the one hand, there is no clear documentation of design decisions taken during the coding phase, making the evolution and the maintenance of the systems difficult. On the other hand, when using specific programming languages, such as C in Microsoft RDS [27], the possibility of generalizing concepts - that could be extracted, reused and applied in different systems - is wasted and the code is written from scratch over and over again.

Thus, currently used methodologies and toolsets are not enough, and it is widely accepted that new approaches should be explored. The goal of our work is to investigate on the current use of modern software engineering techniques for developing robotic systems and their actual automation level. Especially, we have explored the OMG standards in this domain [32] and as a consequence we have delineated a methodology for the construction of robotic software systems, taking advantage of the application of the model-driven approach MDA and the OMG robotic specifications, in particular the RTC proposal.

The rest of the paper is organized as follows. Section 2 summarizes the most relevant software engineering techniques for developing robots. Section 3 presents our guidelines for the construction of robotic systems, applying the MDA approach together with the OMG robotic specifications, through a simple case study. Section 4 discusses a set of related works. Finally, conclusions are presented in Sect. 5.

2 Software Engineering Techniques for Developing Robots

Although the complexity of robotic software is high, in most cases reuse is still restricted to the level of libraries. At the lowest level, a multitude of libraries have been created for robot systems to perform tasks like mathematical computations for kinematics, dynamics and machine vision [14]. Instead of composing systems out of building blocks with assured services, the overall software integration process for another robotic system often is still re-implementation of the glue logic to bring together the various libraries. Often, the kind of overall integration is completely driven by a certain middleware system and its capabilities. This is not only expensive and wastes tremendous resources of highly skilled roboticists, but this also does not take advantage from a maturing process to enhance overall robustness.

From this perspective, it is widely accepted that new approaches should be established to meet the needs of the development process of today's complex robotic systems. Component-based development (CBD) [45], Service Oriented Architecture (SOA) [10], as well as Model Driven Architecture (MDA) [31] are among the key promising technologies in the robotic systems domain. These technologies have been adopted by the Robotics Domain Task Force (RTF) [32], which promotes the integration of modular robotic systems components through the use of OMG standards.

In first place, the CBD paradigm states that application development should be achieved by linking independent parts, the components. Strict component interfaces based on predefined interaction patterns decouple the sphere of influence and thus diminishing 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 services provided and the services required by a given component and defining a composition theory based on the notion of connector. Component technology offer high rates of reusability, but little flexibility with regard to the implementation platform: most existing components are linked to C/C++ and Linux, e.g. Microsoft robotics developer studio [27], EasyLab [7], Player/Stage project [20]. On the other hand, some proposals achieve more independence, thanks to the use of some middleware, e.g. Smart Software Component model [43], Orocos [14], Orca [12] and CLARAty [29].

In second place, SOA is a flexible set of design principles used during the phases of systems development and integration. SOA separates functions into distinct units, or services which developers make accessible over a network in order to allow designers to combine and reuse them in the production of applications. These services and their corresponding consumers communicate with each other by passing data in a well-defined, shared format.

Finally, the MDE [44] approach has emerged as a paradigm shift from code-centric software development to model-based development. Such approach promotes the systematization and automation of the construction of software artifacts. Models are considered as first-class constructs in software development, and developers' knowledge is encapsulated by means of model transformations. Models are implementation-independent and they are automatically transformed to executable code. The MDA is the OMG realization of the MDE. The MDA process can be divided into three phases: the first phase builds a PIM, which is a high-level technology-independent model; then, the previous model is transformed into one or more PSMs; these models are lower level and describe the system in accordance with a given deployment technology; finally, the source code is generated from each PSM.

3 OMG Standards for Robotic Components

The Object Management Group (OMG) is an international, open membership, not-for-profit technology standards consortium. OMG Task Forces develop enterprise integration standards for a wide range of technologies and industries. OMG modeling standards enable visual design, execution and maintenance of software and other processes. Originally aimed at standardizing distributed object-oriented systems, the company now focuses on modeling (programs, systems and business processes) and model-based standards. OMG evolved towards modeling standards by creating the standard for the Unified Modeling Language (UML) followed by related standards for Model Driven Architecture (MDA).

Specifically in the area of Robotics, in 2005 the OMG launched the Robotics Domain Task Force (RTF) with the purpose of fostering the integration of robotics systems from modular components through the adoption of OMG standards. To realize this purpose, the RTF has been promoting important actions and in the last years has released a set of specifications: Robotic Technology Component (RTC) [35], Robotic Interaction Service

(ROIs) [36], Dynamic Deployment and Configuration for Robotic Technology Component (DDC4RTC), Unified Component Model for Distributed, Real-time and Embedded Systems (UCM), Finite State Machine Component for RTC (FSM4RTC) [33], Hardware Abstraction Layer for Robots (HAL4RT) [34], among others.

Let's slightly describe some of these standards:

The RTC proposal specifies a component model that meets the requirements of robotic systems. A component in RTC is a logical representation of a hardware and/or software entity that provides well-known functionality and services. So, the developers can combine RTCs from multiple vendors into a single application, allowing them to create more flexible designs more quickly than before. It includes a Platform-Independent Model (PIM) expressed in UML and three Platform-Specific Models (PSMs) expressed in OMG IDL: Local, Lightweight CMM and CORBA. In the Local PSM, the components reside on the same network node and communicate over direct object references without the mediation of a network or network-centric middleware such as CORBA. In the Lightweight CMM, most components are assumed to be distributed relative to one another and they communicate using a CMM-based middleware. And in CORBA, components are also assumed to be distributed and they communicate using a CORBA-based middleware.

The RoIS Framework abstracts the hardware in the service robot (sensors and actuators) and the Human-Robot Interaction (HRI) functions provided by the robot. It provides a uniform interface between the service robot and the application. Using the RoIS Framework as an intermediary, a service application selects and uses only necessary functions and leaves hardware-related matters, such as which sensor to use, to the HRI engine.

The DDC4RTC specification defines data models and service interfaces of deployment and configuration for RTC based dynamic applications as an extension to DEPL (OMG Deployment and Configuration of Component-based Distributed Applications Specification) specification. Generally speaking, since system structure and configuration are frequently affected by robot movement and application or scenario state, it is important to be able to represent and realize dynamic component deployment and runtime re-configuration requirements.

The HAL4RT specification defines the Platform-Independent Model (PIM) of a Hardware Abstraction Layer for robotic systems that is capable to support at least the following devices: Sensors (sensor kind and unit of measure should be provided) and Actuators (commands to perform motions, and motion feedback information should be provided). In addition this specification defines the Platform specific Model (PSM) in language C based on the HAL PIM. This specification aims to enable engineers such as device makers, device users, and software users to build robotic software without any concern about the differences among the targeted devices, by standardizing the API of these devices.

All these standards interact with each other to provide a higher level of abstraction that facilitates the task of programming robots.

4 A Case Study: Applying MDA with the OMG Robotic Standards

In this section we describe the development of a robotic system applying the MDA approach. The code is automatically derived from models compliant with the OMG robotic standards. For implementing this case study we use the modeling tool Papyrus [37], an Eclipse graphical editing tool for UML2. In accordance with its primary goal of implementing the complete UML2 standard specification, Papyrus provides an extensive support for UML profiles. It includes facilities for defining and applying UML profiles in a very rich and efficient manner. But, it also provides powerful tool customization capabilities similar to DSML-like meta-tools. In this way, Papyrus is a tool that brings together the advantages of using a general purpose language such as UML2, and those of DSML-based approaches. In particular, the SOA and RTC profiles are smoothly incorporated to Papyrus. On the other hand, for implementing the transformations we use Acceleo [19], an open source code generator implementing the OMG's MOF Model to Text Language (MTL) standard that uses any EMF based models (e.g., UML, SysML) to generate any kind of code (e.g., Java, C, PHP) while keeping the traceability of the generated text.

4.1 Using the RTC Standard

The purpose of the RTC specification is to manage the lifecycle of all components in a uniform way. This specification does not attempt to replace the existing UML

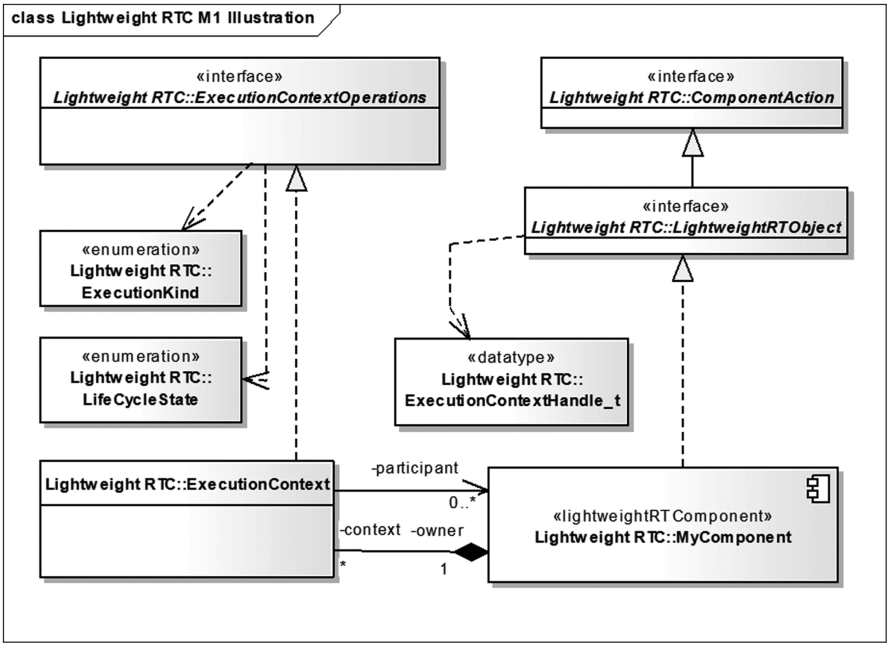

Fig. 1. Simplified LightweightRTC metamodel definition

component models, but focuses on structural and behavioral characteristics required by robotic applications that are not covered by other UML models. It also separates functional specification and execution control. By extending the general-purpose component functionality of UML with direct support for domain-specific structural and behavioral design patterns, RTC elements can serve as powerful building blocks in a robotics system.

The RTC PIM consists of three parts: The Lightweight RTC, the Execution semantics and the Introspection, as follows,

The Lightweight RTC describes a simple model containing definitions of concepts such as component, port and similar ones.

The Execution semantics are extensions to Lightweight RTC to directly support critical design patterns used in robotics applications such as periodic sampled data processing, discrete event/stimulus response processing and modes of operation.

And finally, the Introspection is an API allowing for the examination of elements at runtime. It is useful for dynamic component networks.

The Lightweight RTC specification (see Fig. 1) defines the stereotype lightweightRTComponent extending UML basic component, and describes some interfaces which enable communication between components. When the stereotype is applied, the component must implement all the methods that were defined in the required interfaces. On the other hand, a RTC component may participate in any number of execution contexts. These contexts shall be represented to a RTC component as instances of ExecutionContext class. The ExecutionContext manages the behavior of each RTC component that participates in it.

4.2 The Robot Firefighter

To illustrate our approach, we use a small example of a mobile robot to fight fires. This robot must move and navigate itself around a platform with random obstacles and must find fire sources. Once a flame is detected, the robot begins navigating towards the flame to extinguish it. To improve the efficiency of the robot in the fire extinction, the robot interacts with pre-existing systems. These systems are not part of the robot, but cooperate with it to fulfill its purpose. On one side there are fire detectors placed physically in the environment at strategic locations. Also a Map Service is available. These devices are accessible as external services on the web. All of these services work together for determining if there is a fire in progress. If so, the robot should navigate towards the flame and extinguish it. Each of these devices covers a monitoring zone. When the device indicates the presence of fire, the robot should ask the Map Service how to get to that area. For this, the robot must provide its own position - which it knows through its GPS - to the Map Service. The Map Service then returns a trajectory that the robot must follow to reach the destination.

In first place, the PIM models for this robotic system should be created. By applying the CBD paradigm, robotic elements, such as actuators and sensors, are modeled as components. Thus, the following components were identified: ObstacleDetector, MotionController, NearByFireDetector, FireExtinguisher, GPS, FireDetector and MapService.

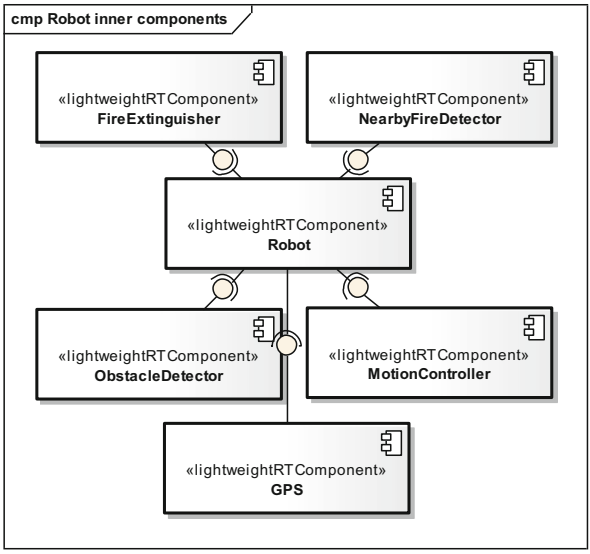

Fig. 2. PIM of the robot firefighter: inner component model.

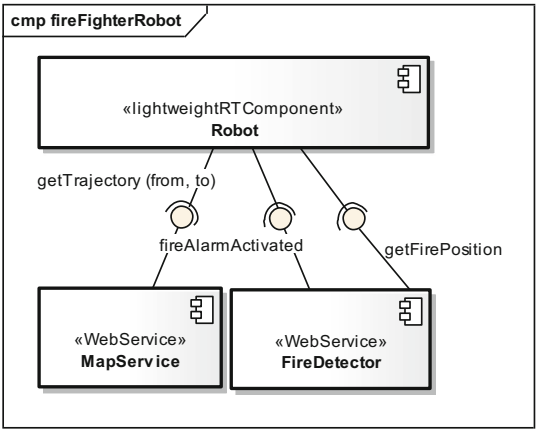

Fig. 3. PIM of the robot firefighter: component and service model.

The first five components are inner components, physically allocated into the robot, while the last two are external components that do not form part of the robot, but collaborate with it by providing helpful services. All of the components provide ports to communicate with each other and they are connected to the robot with their respective glue code. Figure 2 shows the composition of the robot, describing its inner components: ObstacleDetector, MotionController, NearByFireDetector, FireExtinguisher and GPS.

These PIM models are expressed in the UML language enriched with the RTC stereotypes. Figure 3 presents the PIM models specifying the external services (i.e., FireDetector and MapService) as components. In our specific case, the service model is reduced to two elements, but in more complex systems, several services can be smoothly modeled.

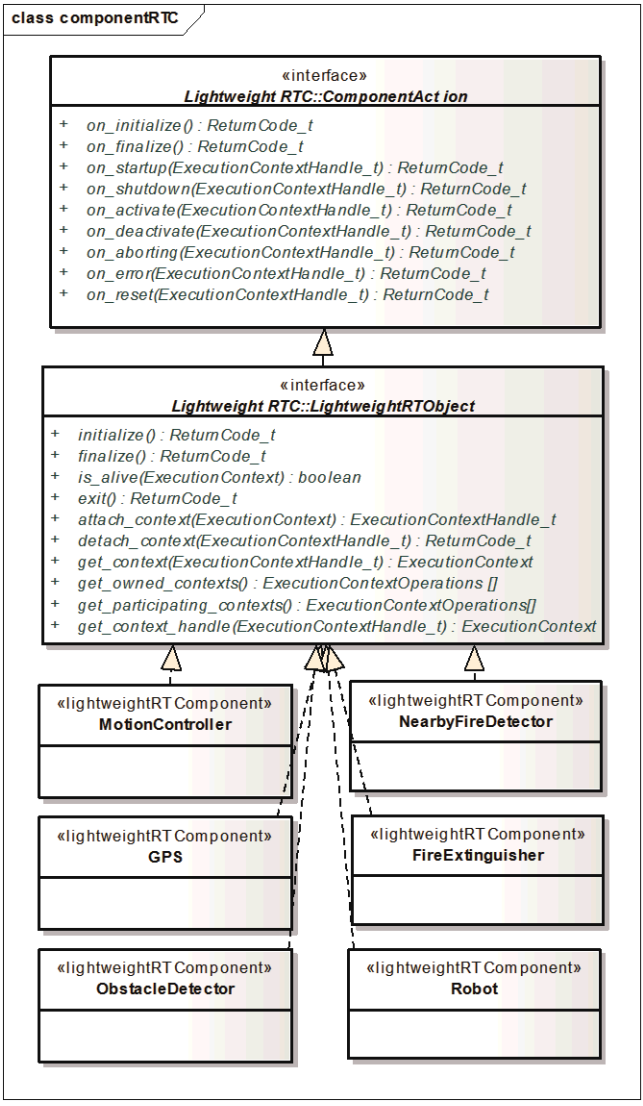

Fig. 4. PSM of the robot firefighter: component's implementation.

These PIM models are expressed in UML language enriched with the RTC stereotypes. Figure 4 shows the PSM model that is automatically derived from the PIM model in Fig. 2. This PSM describes the design of the system complying with the RTC specification. The interface LightweightRTObject defines a lifecycle standard, specifying the states and transitions through which all RTCs will pass from the time they are created until the time they are destroyed. The ComponentAction interface provides callbacks corresponding to the execution of the lifecycle operations of LightweightRTObject. A RTC developer may implement these callback operations in order to execute application-specific logic pointing response to those transitions.

Once the structural models are stable, the behavioral models describing the interaction among components are created. Figure 5 shows a UML state machine describing the overall behavior of the robot. The state machine specifies the four states which the robot can go through: walkAround, navigatingTowardsTheFirePosition, approachingTheFlame and fireExtinguish. Immediately after starting its workflow the robot enters to the state walkAround, and remains in the same state while no fire is detected. When the fire detector triggers an alarm the robot switches to the state navigatingTowardsTheFirePosition. Then, the robot keeps in the same state, moving in the direction of the fire, until the fire is reached.

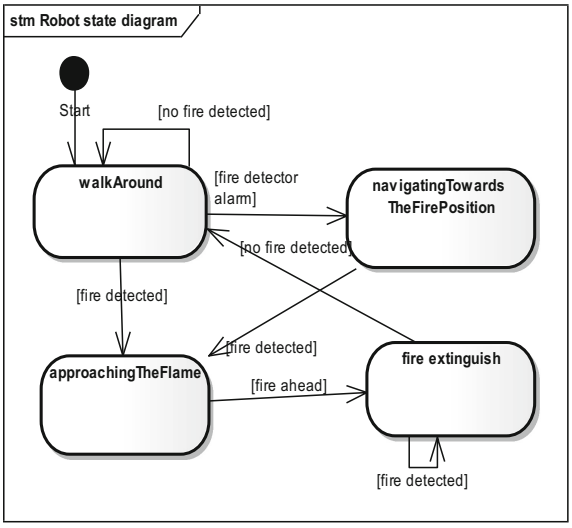

Fig. 5. PIM of the robot firefighter: overall behavioral model

Once the robot reaches the fire it enters to the state approachingTheFlame. In such state the robot approaches the fire as close as possible. When the fire is very close the robot switches to state fireExtinguish where it triggers mechanisms to extinguish the fire.

Other behavioral models are created for the remaining behaviors of the robot, but are not presented in this paper for space limitations.

Then, similarly to what was done with the structural models, PSM behavioral models are automatically derived. For example, Fig. 6 shows a PSM of the robot´s behavior that was automatically derived from the State machine in Fig. 5 by applying the state pattern as prescribed by the RTC.

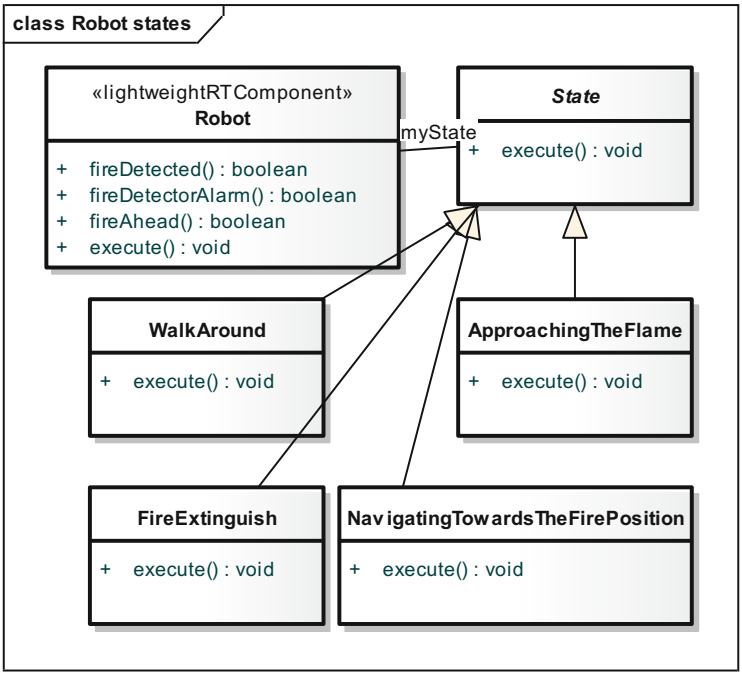

Fig. 6. PIM of the robot firefighter: behavior's implementation.

For each state in the state machine, a class is created as subclass of the abstract class State. Each transition trigger in the state machine is mapped to a Boolean operation in the main class Robot. A method named execute() is defined in the class Robot; according to the State Pattern this method just delegates its behavior to the execute() method in the corresponding State.

The next step of the development process consists in the transformation of structural as well as behavioral PSM models to a specific programming language. The following listing shows the transformation program written in Acceleo that takes as input the robot structural models and produces Java code as output.

```
[module generateRTCCustomComponents
('http://www.eclipse.org/uml2/5.0.0/UML')]

[template public generateElement(aComponent : Component)]
[comment @main/]
[file ('myRTC/' +aComponent.name+'.java',false, 'UTF-8')]

package myRTC;
import java.util.List;
import java.util.ArrayList;
import lightweightRTC.ExecutionContext;
import lightweightRTC.ExecutionContextHandle_t;
import lightweightRTC.ExecutionContextOperations;
import lightweightRTC.LightweightRTObject;
import lightweightRTC.ReturnCode_t;

public class [aComponent.name/] implements LightweightRTObject {

private List<ExecutionContext> contexts= new ArrayList();
  /*
  * @generated
  */
  public ReturnCode_t finalize_() {
    return this.on_finalize();
  }
```

The acceleo program above generates the following Java code as output,

```
package myRTC;

import java.util.List;
import java.util.ArrayList;
import lightweightRTC.ExecutionContext;
import lightweightRTC.ExecutionContextHandle_t;
import lightweightRTC.ExecutionContextOperations;
import lightweightRTC.LifeCycleState;
import lightweightRTC.LightweightRTObject;
import lightweightRTC.ReturnCode_t;

public class Robot implements LightweightRTObject {
  private List<ExecutionContext> contexts = new ArrayList();
  /*
  * @generated
  */
  public ReturnCode_t finalize_() {
    return this.on_finalize();
  }
```

The rationale for building this Java program was the following, for each component in the PIM, a Java class was created as an implementation of the LightweightRTObject standard interface. Additionally all the Lightweight RTC resources were imported in the program.

4.3 Lessons Learned from the Case Study

In this case study we have identified the different models that can be created to specify both the structure and the behavior of the robot. These models were represented using the OMG robotic standard, which is basically the well-known UML language enriched with appropriate stereotypes to describe structural and behavioral characteristics required by robotic applications that are not covered by other UML models. This standard specification manages the lifecycle of all robotic components in a uniform way. Additionally, the case study shows how the models are gradually defined at different abstraction levels, starting with the more abstract models, completely independent of the platform, from which other less abstract models could be automatically derived, to finally get to the executable code.

5 Related Work

It is broadly recognized that there is a need to incorporate software engineering principles within the development of future robot platforms. This has lead in the last years to the conception of a set of activities with the objective of assembling researchers from both fields, Model-Driven Software Development on one hand and Robotics on the other hand. Examples of these activities are the International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob) [42] launched in 2009, and the Workshop on Model-Driven Robot Software Engineering(MORSE) [3] initiated in 2013, both with the goal of incentivizing the interaction of these areas. As a result, in recent years several software frameworks have been developed to provide simple and intuitive ways of writing software applications for robot platforms. This includes academic research as well as industrial products.

On the industrial side one of the most well known is Lego Mindstorms Evolution 3 [26], developed especially for the Lego robots which can be built out of the Lego model kits. This is an extremely flexible and powerful system which allows anyone to build a robot using a few standard parts like motors, color sensors, touch sensors, infrared sensors and other Lego elements. Afterwards, the user can graphically implement a program by choosing the desired activities from the pallet of available blocks. Because of this target group, the software only has a limited set of functions and cannot be extended in any way. Evolution 3 only supports the creation of software for Lego robots, and thus cannot be regarded as a general robot modeling framework.

Other industrial tool is Choregraphe [1], an environment developed by Aldebaran Robotics, the manufacturer of the NAO humanoid robot, to allow robots to be programmed by graphical applications. It also supports code reuse and debugging

capabilities and makes it possible to monitor and control NAO robots manually. The program uses an intuitive drag-and-drop interface in which a program is created using boxes that can be combined into a kind of flow diagram. In summary, although it is easy to use, Choregraphe allows the creation of complex programs. Like Lego Mindstorms Evolution 3, Choregraphe can only be used in combination with the NAO robot and thus cannot be regarded as a general robot modeling framework.

Another example of industrial product is Robotino View 2, a visual development environment provided by Festo Didactic exclusively for Robotino robots. Robotino View 2 shares the same limitation as the two previously mentioned frameworks — it is proprietary and can only be used with one kind of robot.

Finally, Microsoft Robotics Developer Studio 4 (MRDS4) [27] is another programming environment for building robotics applications. It provides a Visual Programming Language with an intuitive drag-and-drop interface for hobbyists and support for Microsoft Visual Studio for professional developers. It has several significant advantages. First, numerous robots are supported. Second, a high-fidelity simulation environment is provided by Visual Simulation Environment (VSE), and the functionality of MRDS4 can be extended by providing additional libraries and services. Also, extensive documentation, samples and tutorials are available.

On the academic side, many works [8, 11, 12, 23, 25, 28, 49] has taken advantage of CBD for developing robotic systems whilst other proposals [4, 16, 18, 49] have applied SOA to building robotic systems. Promising proposals were found for applying model-driven development to robotics [2, 5, 6, 9, 13, 17, 21, 22, 24, 25, 39–41, 46, 48], while only one work combined all three technologies [47]. Let us examine the most representative ones:

Atkinson and colleagues in [2] introduce a prototype domain-specific modeling framework designed to support the quick, simple and reliable creation of control software for standard robot platforms. In this paper they have presented a prototype framework, known as the Deep Robot Modeling Framework (DRMF). The current version of the prototype supports a rudimentary implementation of all of these features in the context of the NAO robot platform developed by Alderbaran Robots, although the basic framework is platform independent. Applications developed using the NAO-specific languages are automatically mapped into C++ code that can be loaded onto, and used to drive, individual NAO robots.

Dhouib and colleagues in [17] define the language RobotML as an extension to the Eclipse-based UML modeling tool Papyrus. Papyrus puts strong emphasis on UML's profile mechanism, which allows domain-specific adaptations. RobotML aims to provide model-driven engineering capabilities for the domain of robot programming, implementing code generators for different target platforms.

In [15] a small and declarative domain-specific language for pick and place applications was elaborated for demonstrating the feasibility of the model driven approach. Configurable code generation for C++ is provided.

These related works focus on defining specific modeling languages that enable the designer to create abstract models of the robotic system and to automatically generate code from them. Although these different languages and platforms are superficially very different, at a high enough level of abstraction they all contain the same basic constructs

– predefined types representing the components and actions from which the structure and behavior of individual robots are constructed. In principle, therefore, they could all be brought together under the umbrella of a single, unified robot modeling framework. We believe that our approach makes a contribution towards the application of standard instead of developing new concepts which are then difficult to integrate.

6 Conclusions

Programming robots is a complicated and time-consuming task. Often, control and communication paths within the system are tightly coupled to the actual physical configuration of the robot. Robotic researchers have been mainly concentrated on creating hardware/software solutions for specialized tasks, leading to an extensive landscape of comparable but isolated solutions which cannot be reused and combined easily. Furthermore, these approaches lack comprehensive software engineering methodologies and abstractions to handle the increased heterogeneity and complexity of robotic software systems.

The contribution of this research consists in delineating guidelines for the construction of robotic software systems, taking advantage of the application of the OMG standard robotic specifications which adhere to the model-driven approach MDA. Model-driven approaches further simplify the reuse of already implemented and tested modules by enabling developers to model their applications on a higher abstraction level incorporating existing modules, managing the complexity and facilitating the reusability of robot code.

We observed that the CBD and SOA paradigms provide a starting point for a MDA 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. In particular, the proposed methodology takes advantages of the standards defined by the Robotics Domain Task Force (RTF) which promotes the integration of modular robotic systems components under the umbrella of MDA.

The approach captures 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 abstraction level.

The proposed methodology has been prototyped using Papyrus and Acceleo that are tools provided by the Eclipse Modeling Project that focuses on the evolution of model-based development technologies within the Eclipse community.

At the moment, there are few proposals taking advantage of the combined application of CBD, SOA and MDA to robotic software system development as reviewed in [38] and more recently in [30], and there is a lack of proposals towards the application of the OMG robotic standard.

References

1. Aldebaran Robotics: Choreographe overview (2014). http://doc.aldebaran.com/2-1/software/choregraphe/index.html
2. Atkinson, C., Gerbig, R., Markert, K., Zrianina, M., Egurnov, A., Kajzar, F.: Towards a deep, domain-specific modeling framework for robot applications. In: Workshop on Model-Driven Robot Software Engineering, MORSE 2014 (2014)
3. Aßmann, U., Atkinson, C., Burger, E., Goldschmidt, T., Reussner, R. (eds.): Proceedings of MORSE/VAO 2015, Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering, Italy. ACM, New York (2015)
4. Amoretti, M., Zanichelli, F., Conte, G.: A service-oriented approach for building autonomic peer-to-peer robot systems In: 16th IEEE International Workshops on, WETICE 2007 (2007)
5. Arney, D., Fischmeister, S., Lee, I., Takashima, Y., Yim, M.: Model-based programming of modular robots. In: 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), pp. 66–74 (2010)
6. Baer, P.A., Reichle, R., Zapf, M., Weise, T., Geihs, K.: A generative approach to the development of autonomous robot software. In: Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems, EASe 2007 (2007)
7. Barner, S., Geisinger, M., Buckl, C., Knoll, A.: EasyLab: model-based development of software for mechatronic systems. In: IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Beijing, China (2008)
8. Basu, A., Bensalem, B., Bozga, M., Combaz, J., Jaber, M., Nguyen, T., Sifakis, J.: Rigorous component-based system design using the BIP framework. IEEE Softw. **28**(3), 41–48 (2011)
9. Baumgartl, J., Buchmann, T., Henrich, D., Westfechtel, B.: Towards easy robot programming: using DSLs, code generators and software product lines. In: Proceedings of the 8th International Joint Conference on Software Technologies, ICSOFT 2013 pp. 548–554 (2013)
10. Bell, M.: Introduction to Service-Oriented Modeling. Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley, Hoboken (2008)
11. Biggs, G.: Flexible, adaptable utility components for component-based robot software. In: 2010 IEEE International Conference on Robotics and Automation (ICRA), pp. 4615–4620 (2010)
12. Brooks, A., Kaupp, T., Makarenko, A., Oreback, A., Williams, S.: Towards component-based robotics. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005) (2005)
13. Brugali, D., Shakhimardanov, A.: Component-based robotic engineering (Part II). IEEE Robot. Autom. Mag. **17**(1), 100–112 (2010)
14. Bruyninckx, H.: Open robot control software: the OROCOS project. In: Proceedings of 2001 IEEE International Conference on Robotics and Automation (ICRA 2001), Korea, vol. 3 (2001)
15. Buchmann, T., Baumgartl, J., Henrich, D., Westfechtel, B.: Towards a domain-specific language for pick-and-place applications (2014). arXiv:1401.1376
16. Cesetti, A., Scotti, C.P., Di Buo, G., Longhi, S.: A service oriented architecture supporting an autonomous mobile robot for industrial applications. In: 18th Mediterranean Conference on Control and Automation (MED), pp. 604–609 (2010)
17. Dhouib, S., Kchir, S., Stinckwich, S., Ziadi, T., Ziane, M.: RobotML, a domain-specific language to design, simulate and deploy robotic applications. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) SIMPAR 2012. LNCS, vol. 7628, pp. 149–160. Springer, Heidelberg (2012)

18. Ebenhofer, G., Bauer, H., Plasch, M., Zambal, S.: A system integration approach for service-oriented robotics. In: 2013 IEEE 18th International Conference on Emerging Technologies and Factory Automation, ETFA 2013, Italy, September 2013
19. Eclipse Acceleo Project. http://www.eclipse.org/acceleo/
20. Gerkey, B.P., Vaughan, R.T., Howard, A.: Most valuable player: a robot device server for distributed control. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2001)
21. Son, H.S., Kim, W.Y., Kim, R.: Semi-automatic software development based on MDE for heterogeneous multi-joint robots. In Future Generation Communication and Networking Symposia, FGCNS 2008, pp. 93–98 (2008)
22. Iborra, A., Caceres, D., Ortiz, F., Franco, J., Palma, P., Alvarez, B.: Design of Service Robots. Experiences Using Software Engineering. IEEE Robotics and Automation Magazine 1070-9932/09/, pp. 24–33. IEEE, March 2009
23. Jawawi, D.N.A., Deris, S., Mamat, R.: Early-life cycle reuse approach for component-based software of autonomous mobile robot system. In: Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing Conference (2008)
24. Jorges, S., Kubczak, C., Pageau, F., Margaria, T.: Model driven design of reliable robot control programs using the jABC. In: Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems, EASe 2007, pp. 137–148 (2007)
25. Jung, E., Kapoor, C., Batory, D.: Automatic code generation for actuator interfacing from a declarative specification. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS 2005), pp. 2839–2844 (2005)
26. LEGO. http://shop.lego.com/en-US/ LEGO-MINDSTORMS-EV3-31313; Visited April 2014
27. Microsoft Robotics Group: Robotics Developer Studio: Reference Platform Design (2012)
28. Jung, M.Y., Deguet, A., Kazanzides, P.: A component-based architecture for flexible integration of robotic systems. In: IEEE/International Conference on Intelligent Robots and Systems (2010)
29. Nesnas, I., Wright, A., Bajracharya, M., Simmons, R., Estlin, T.: CLARAty and challenges of developing interoperable robotic software. In: Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), vol. 3, pp. 2428–2435 (2003)
30. Nordmann, A., Hochgeschwender, N., Wigand, D.L., Wrede, S.: A survey on domain-specific modeling and languages in robotics. J. Softw. Eng. Robot. 7(1), 75–99 (2016)
31. OMG MDA Guide 2.0. http://www.omg.org/mda. Edited in Boston on 18 June 2014
32. OMG Robotics Domain Task Force (DTF). http://robotics.omg.org/. Accessed 3 Mar 2016
33. OMG Finite State Machine Component for RTC™ Formal Version Of FSM4RTC™ (2016). http://www.omg.org/spec/FSM4RTC/Current
34. OMG Hardware Abstraction Layer for Robotic Technology™ (HAL4RT™), January 2016. http://www.omg.org/spec/HAL4RT/Current
35. OMG Robotic Technology Component™ (RTC) (2012). http://www.omg.org/spec/RTC/
36. OMG Robotic Interaction Service™ (RoIS™) (2016). http://www.omg.org/spec/RoIS/
37. Papyrus Eclipse project (2015). http://www.eclipse.org/papyrus/
38. Pons, C., Giandini, R., Arévalo, G.: A systematic review of applying modern software engineering techniques to developing robotic systems. Ingeniería e Investigación **32**(1), 58–63 (2012)
39. Poppa, F., Zimmer, U.: RobotUI - a software architecture for modular robotics user interface frameworks. In: 25th IEEE/RSJ International Conference on Robotics and Intelligent Systems, IROS 2012, Algarve, Portugal, pp. 2571–2576, October 2012

40. Sanchez, P., Alonso, D., Rosique, F., Alvarez, B., Pastor, J.: Introducing safety requirements traceability support in model-driven development of robotic applications. IEEE Trans. Comput. **60**, 1059–1071 (2010)
41. Schlegel, C., Steck, A., Lotz, A.: Robotic software systems: from code-driven to model-driven software development (Chapter 23). In: Robotic Systems - Applications, Control and Programming (2012). ISBN 978-953-307-941-7
42. Schlegel, C., Schultz, U., Stinckwich, S., Wrede, S.: Proceedings of the Sixth International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob 2015) (2015). arXiv:1601.00877
43. SmartSoft, July 2013. http://smart-robotics.sourceforge.net/
44. Stahl, T., Voelter, M.: Model Driven Software Development. Wiley, Hoboken (2006). ISBN 0470025700
45. Szyperski, C.: Component Software: Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley Professional, Boston (2002). ISBN 0-201-74572-0
46. Thomas, U., Hirzinger, G., Rumpe, B., Schulze, C., Wortmann, A.: A new skill based robot programming language using UML/P Statecharts. In: Proceedings - IEEE International Conference on Robotics and Automation, ICRA 2013, Germany, pp. 461–466, May 2013
47. Tsai, W.T., Huang, Q., Sun, X.: A collaborative service-oriented simulation framework with microsoft robotic studio®. In: 41st Annual Simulation Symposium, ANSS 2008 (2008)
48. Wei, H., Duan, X., Li, S., Tong, G., Wang, T.: A component based design framework for robot software architecture. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3429–3434 (2009)
49. Yang, T.-H., Lee, W.-P.: A service-oriented framework for the development of home robots. Int. J. Adv. Robot. Syst. **10**(122) (2013)