A Model-Driven Approach to Constructing Robotic Systems

Claudia Pons[1,2,3] , Gabriela Pérez[1] , Roxana Giandini[1], Gabriel Baum[1,2]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata
[2] CIC, Comisión de Investigaciones Científicas PBA
[3] UAI, Universidad Abierta Interamericana
Buenos Aires, Argentina

{gperez, cpons, giandini, gbaum}@lifia.info.unlp.edu.ar

ABSTRACT

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 work 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.

Keywords: robotic software system, software development process, software engineering, model driven development

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. Additionally, this special kind of systems needs more quality than a general purpose system and it has to be able to cope with the uncertain and dynamic physical environment where they are immerse. Attributes like reliability and safety are a strong requirement in this domain.

Furthermore, robotic systems consist of different hardware components and different sensors which results in very complex and highly variable system architecture. 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 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 that 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 (Gira, 2013), or C in Microsoft RDS (Microsoft, 2009), 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 traditional development approaches are reaching their limits; currently used methodologies and toolsets fall short to address the needs of such complex systems. In this context, 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) (Szyperski, 2002), Service Oriented Architecture (SOA) (Bell 2008 and 2010), as well as Model Driven software Engineering (MDE) (Stahl, 2006), (Pons et al., 2010) and Domain-Specific Modeling (DSM) (Steven and Juha-Pekka, 2008) are among the key promising technologies in the Robotic systems domain.

In our project, we 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. Our objective 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. WORKING METHODOLOGY

In this context, it is mandatory to work towards applying engineering principles to cope with the complexity of robotic software systems because we cannot expect significant growth with hand-crafted single-unit systems. On the other hand, 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 technological platforms. Separating robotics knowledge from short-cycled implementation technologies is essential to foster reuse and maintenance. Thus, applying existing software engineering technology, such as MDE, SOA and CBD, to building robotic software systems would save a great amount of time and effort while favor reusability of such systems. Within this background, the specific outcomes of this project are:

- Summarizing the existing evidence concerning the application of software engineering technologies such as SOA, MDE and CBD on the robotic systems development field;
- Identifying gaps in current research in order to suggest areas for further investigation;
- Providing a background in order to appropriately position new research activities;
- Building on the application of modern techniques providing an advance in the field;
- Defining an open methodology for the robotic development process.
- 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, component definition editors, etc.
- Providing technological and methodological tools with highly dynamic adaptive capacity to cope with the rate of development of robotics and the local differences of each hardware platform.
- Performing a series of experiments to assess the effectiveness and feasibility of the proposal in the construction of complex robotic systems.

3. EXISTING APPROACHES

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, such as (Bruyninckx, 2001). Instead of composing systems out of building blocks with assured 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 kind of overall integration is completely driven by a certain middleware system and its capabilities. Obviously, 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. We have faced this problem in our own practice. We have been programming educational robots for more than 10 years (GIRA, 2013) (CAETI, 2013) and we have observed in the last years the emergence of robotic kits oriented to non-expert users that 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 encountered 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 over and over again. An example of these interfaces is "Physical EToys" (GIRA, 2013) that proposes a standard teaching platform for programming robots, regardless of whether they are based on Arduino, Lego, or other technologies.

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) (Szyperski, 2002), Service Oriented Architecture (SOA) (Bell 2008 and 2010), as well as Model Driven software Engineering (MDE) (Stahl, 2006), (Pons et al., 2010) and Domain-Specific Modeling (DSM) (Steven and Juha-Pekka, 2008) are among the key promising technologies in the Robotic systems domain.

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 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 services provided and the services required 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 with regard to the implementation platform: most existing component are linked to C/ C++ and Linux (e.g. Microsoft robotics developer studio (Microsoft, 2009), EasyLab (Barner et al., 2008), Player/Stage project (Gerkey et al., 2001)), although some achieve more independence, thanks to the use of some middleware (e.g. Smart Software Component model (smartSoft, 2013), Orocos (Bruyninckx, 2001) Orca (Brooks et al., 2005), CLARAty (Nesnas et al., 2003)).

In second place, we need a way 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 mean 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 MDE and DSM approaches have emerged as a paradigm shift from code-centric software development to model-based development. Such approaches promote 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. The essential characteristic of MDE and DSM is that software development's 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. This is especially relevant for software systems within the ubiquitous computing domain, which consist of dynamic, distributed applications and heterogeneous hardware platforms, such as robotic systems.

Finally, SOA 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. SOA separates functions into distinct units, or services (Bell, 2008) which developers make accessible over a network in order to allow users 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 (Bell, 2010).

Summarizing, a growing tendency was identified regarding applying component-based development as well as service-based architecture and model-driven software development, although such techniques have mostly been applied in isolation. Some work (Basu et al., 2011; Biggs, 2010; Brooks et al., 2005; Jawawi et al., 2008; Min Yang Jung et al., 2010) has taken advantage of CBD for developing robotic systems whilst other proposals (Amoretti et al., 2007; Cesetti et al., 2010; Ebenhofer et al, 2013; Yang et al, 2013) have applied SOA to building robotic systems. Only preliminary proposals were found for applying model-driven development to robotics (Arney et al., 2010; Baer et al., 2007; Baumgartl et al, 2013; Brugali and Scandurra,2009; Brugali and Shakhimardanov, 2010; Dhouib et al., 2012; Hyun Seung Son et al., 2008; Iborra et al., 2009; Jorges et al., 2007; Jung et al., 2005; Poppa et al, 2012; Sanchez et al., 2010; Schlegel, 2012; Thomas et al, 2013; Wei et al., 2009) while only one work combined all three technologies (Tsai et al., 2008).

4. 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 describe 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 code from models expressed in a standard modeling language.

For using the MDD approach we take advantage of standards defined by the Robotics Domain Task Force (RTF) (OMG, 2013) which promotes the integration of modular robotic systems components through the adoption of OMG standards. Currently, the OMG has released four specifications: Robotic Interaction Service (ROIs), Robotic Localization Service (RLS), Robotic Technology Component (RTC) and Dynamic Deployment and Configuration for Robotic Technology Component (DDC4RTC). Other specifications like Unified Component Model for Distributed, Real-time and Embedded Systems (UCM), Finite State Machine Component for RTC (FSM4RTC), Hardware Abstraction Layer for Robots, among others, are in progress.

The RTC defines a component model and certain important infrastructure services applicable to the domain of robotics software development. It includes a Platform-Independent Model (PIM) expressed in UML and Platform-Specific Models (PSMs) expressed in OMG IDL. A RTC is a logical representation of a hardware and/or software entity that provides well-known functionality and services. By extending the general-purpose component functionality of UML with direct support for domain-specific structural and behavioral design patterns, RTCs can serve as powerful building blocks in an RT system. Developers can combine RTCs from multiple vendors into a single application, allowing them to create more flexible designs more quickly than before. Its goal is a greater compatibility and reusability amongst vendors of robot software, not just the software itself but also the tools. It provides rich component lifecycle to enforce state coherency among components and defines data structures for describing components and other elements. It supports fundamental design patterns, such as Collaboration of fine-grained components tightly coupled in time, Stimulus- response with finite state machines, and Dynamic composition of components collaborating synchronously or asynchronously, among others.

The Robotic Interaction Service (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 a go-between, a service application selects and uses only necessary functions and leaves hardware-related matters, such as which sensor to use, to the HRI engine.

Finally 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 run-time re-configuration requirements.

To illustrate our approach, we use a small example of a 3-wheel robot to fight fires. This robot must move and navigate itself around an enclosed platform with random obstacles and must find fires (ie. lit candles). Once a flame is detected on one of the robots photo sensors, the robot begins navigating towards the flame to extinguish it. The robot is composed of two motors A and B, 3 ultrasonic ranging HC-SR04 modules (to enable the robot to determine its distance from any obstacle), 3 phototransistors (phototransistors are most sensitive to infrared light, making them an appropriate choice for detecting a flame) and a fan (to extinguish the flame).

To improve the efficiency of the robot in the fire extinction, the robot will interact with pre-existing systems. These systems are not part of the robot, but cooperate with it to fulfill its purpose. On one side we have fire detectors placed physically in the environment at strategic locations. These devices are available as external services and are accessed over the network. All of these services will be pooled to determine if there is a fire in progress. If so, the robot should navigate towards the flame and turn it off. Each of these devices covers a monitoring zone. When the device indicates fire, the robot should ask the service map how to get to that area. For this, the robot must provide the service map its own position, which it knows through the GPS. The map service will then return a path that the robot must follow to reach the destination.

Thus, in our example we identify the following inner components: Robot, DistanceSensor, MotionController, FireSensor, FunController and GPS; and the outer components: FireDetector and MapService. So it is worth distinguishing two models: the Component Model showing the internal components, and the Service Model describing the external components. Figure 1 shows the Component and the Service Models together. In our specific case, our service model is reduced to two components. In more complex platforms, we can have several services that can be modeled with their respective glue code to be connected to the implemented robots. Figure 2 presents a UML state machine describing the behavior of the robot.

There are different ways we implement this Robot firefighter. Figure 3 shows the design of the system complying with the RT-Component specification. The interface LightweightRTObject defines a lifecycle standart. It defines 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. An RTC developer may implement these callback operations in order to execute application-specific logic pointing response to those transitions. Figure 4 shows the behavior implementation of the robot.

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 consists in building a PIM that allows abstracting the domain concepts and their functionalities using MDD and CBD. With the models we can then derive the code in any specific robotic language.

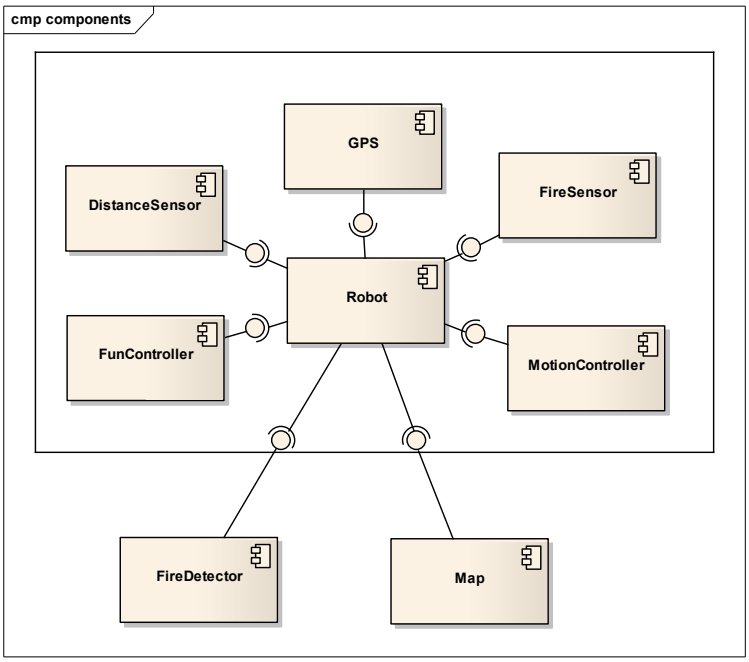

Figure 1. PIM of the robot firefighter: Component Model.

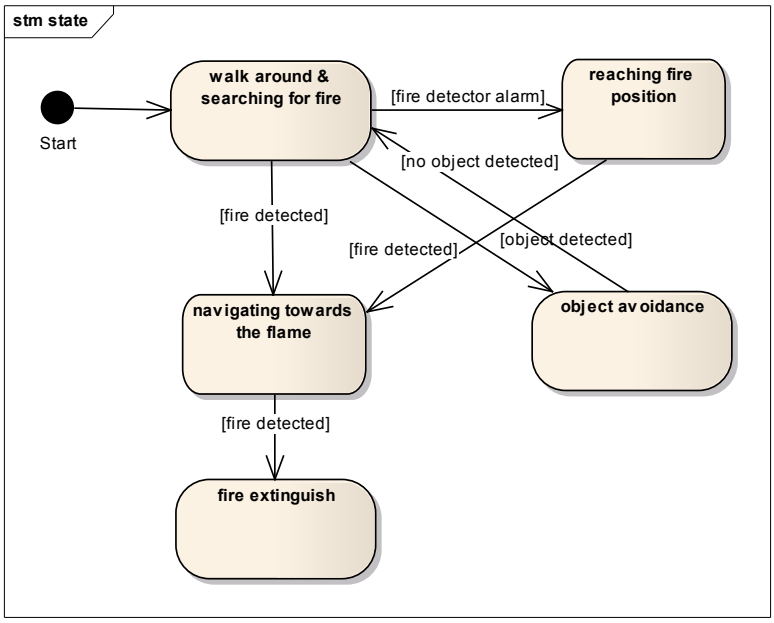

Figure 2. PIM of the robot firefighter: Behavioral Model.

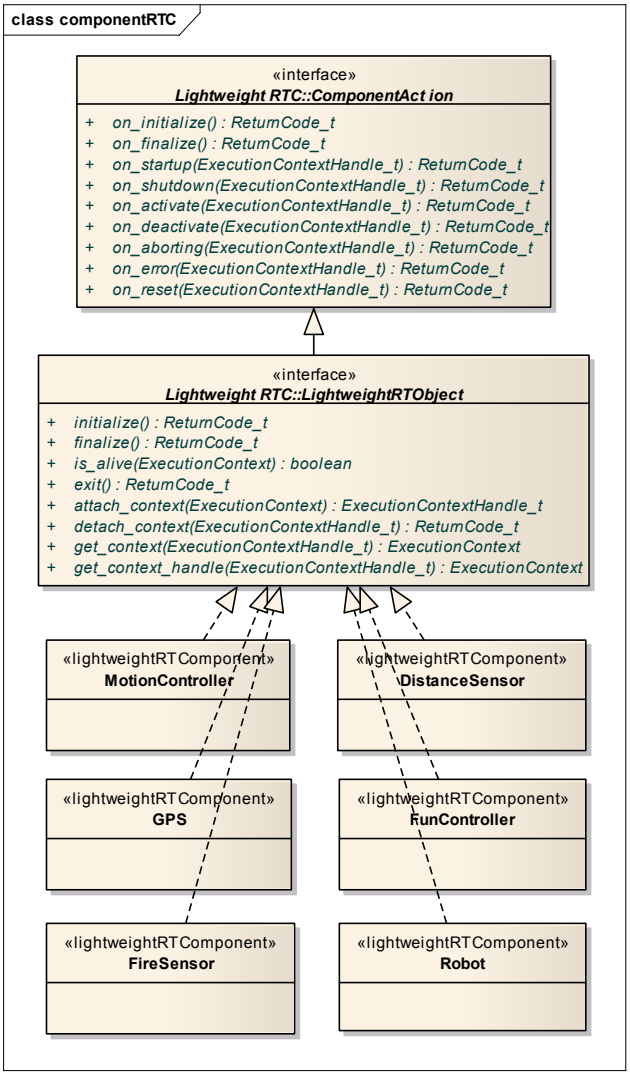

Figure 3. PSM of the Robot firefighter: Component´s implementation.

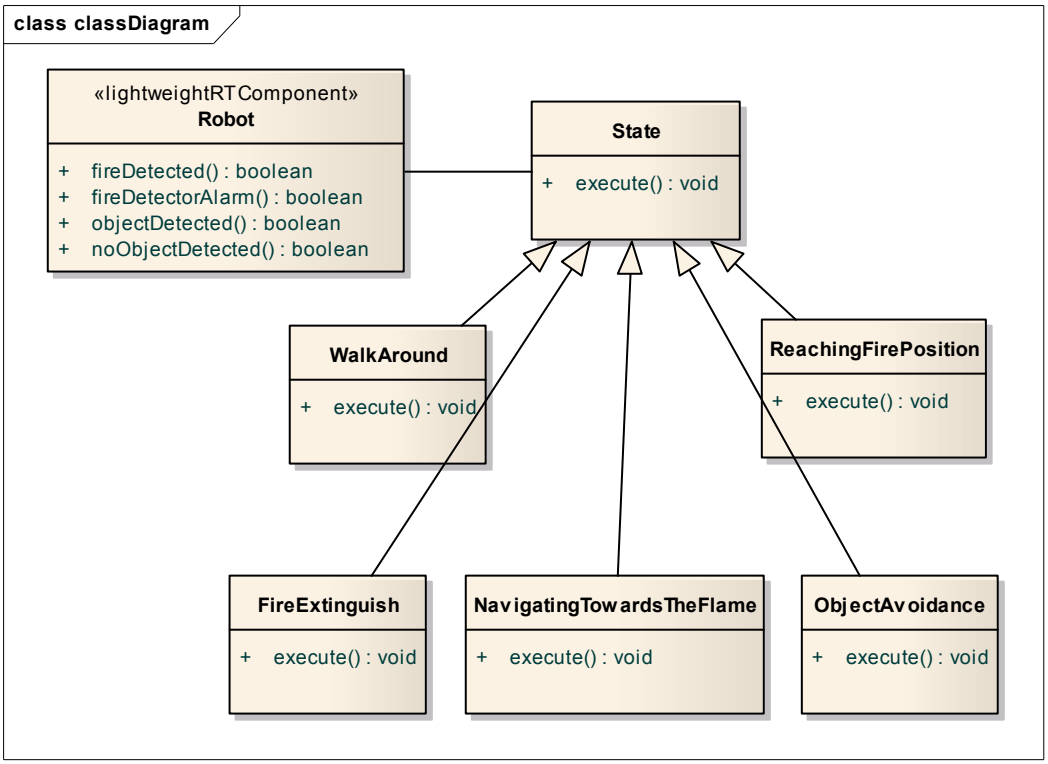

Figure 4. PSM of the Robot firefighter: Behavior´s implementation.

5. 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. Traditional approaches, based on mainly coding the applications without using modeling 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.

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. The contribution of our work consists in the development of the basis for a methodological framework supported with different tools for the construction of robotic software systems using mainly MDD. We observed that the CBD 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. We capture 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.

The proposed methodology has been prototyped and evaluated, and the results show that it can be used to build robotic systems successfully. At the moment, there is no proposal taking advantage of the combined application of CBD, SOA and MDE to robotic software system development as reviewed in (Pons et al., 2012).

6. REFERENCES

Amoretti, M.; Zanichelli, F.; Conte, G.; A Service-Oriented Approach for Building Autonomic Peer-to-Peer Robot Systems Enabling Technologies: Infrastructure for Collaborative Enterprises, 2007. WETICE 2007. 16th IEEE International Workshops on, Page(s): 137 - 142 (2007)

Arney, D.; Fischmeister, S.; Lee, I.; Takashima, Y.; Yim, M.;Model-Based Programming of Modular Robots. 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), Page(s): 66 – 74 (2010)

Baer, P. A.; Reichle, R.; Zapf, M.; Weise,T.; Geihs, K.; A Generative Approach to the Development of Autonomous Robot Software.EASe '07. Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems. (2007)

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)

Basu, A.; Bensalem, B.; Bozga, M.; Combaz, J.; Jaber, M.; Nguyen, T.; Sifakis, J.; Rigorous Component-Based System Design Using the BIP Framework Software, IEEE Volume: 28 , Issue: 3 Page(s): 41 – 48. (2011)

Baumgartl, J., Buchmann,T., Henrich,D., Westfechtel,B.: Towards easy robot programming: Using DSLs, code generators and software product lines ICSOFT 2013 - Proceedings of the 8th International Joint Conference on Software Technologies 2013, Reykjavik; Iceland; Pages 548-554.. ISBN: 978-989856568-6; (July 2013)

Bell, M., "Introduction to Service-Oriented Modeling". Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley & Sons. pp. 3. ISBN 978-0-470-14111-3. (2008).

Bell, M., SOA Modeling Patterns for Service-Oriented Discovery and Analysis. Wiley & Sons. pp. 390. ISBN 978-0470481974. (2010)

Biggs, G.; Flexible, adaptable utility components for component-based robot software. Robotics and Automation (ICRA), 2010 IEEE International Conference on, Page(s): 4615 – 4620. (2010)

Brooks, A., Kaupp, T., Makarenko, A., Oreback, A., Williams, S.: Towards component-based robotics. In: Proc. of 2005 IEEE/RSJ Int. Conf. on Intellegent Robots and Systems (IROS'05), pp. 163–168. (2005)

Brugali, D.; Scandurra, P.; Component-based robotic engineering (Part I) [Tutorial] Robotics & Automation Magazine, IEEE Volume: 16 , Issue: 4, Page(s): 84 - 96 (2009)

Brugali, D.; Shakhimardanov, A.; Component-Based Robotic Engineering (Part II) Robotics & Automation Magazine, IEEE Volume: 17 , Issue: 1 , Page(s): 100 – 112. (2010)

Bruyninckx, H., Open robot control software: The OROCOS project. In: Proceedings of 2001 IEEE International Conference on Robotics and Automation (ICRA'01), vol. 3, pp. 2523–2528. Korea (2001).

CAETI (Centro de Altos Estudios en Tecnología Informática). Robotic projects. http://www.caeti.uai.edu.ar. Access November 1st. 2013.

Cesetti, A.; Scotti, C. P.; Di Buo, G.; Longhi, S.; A Service Oriented Architecture supporting an autonomous mobile robot for industrial applications Control & Automation (MED), 18th Mediterranean Conference on,Page(s): 604 – 609. (2010)

Dhouib, S., Kchir, S., Stinckwich, S., Ziadi, T., and Ziane,M. RobotMLl, "A domain-specific language to design, simulate and deploy robotic applications". In Noda, I., Ando, N., Brugali, D., and Kuffner, J., editors, Simulation, Modeling, and Programming for Autonomous Robots, vol. 7628 of Lecture Notes in Computer Science, pages 149–160. Springer(2012).

Ebenhofer, G., Bauer, H., Plasch, M., Zambal, S., Akkaladevi, S.C., Pichler, A.: A system integration approach for service-oriented robotics. 2013 IEEE 18th International Conference on Emerging Technologies and Factory Automation, ETFA 2013; Cagliari; Italy; ISSN: 19460740; (September 2013).

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, pp. 1226–1231. Wailea, Hawaii (2001)

GIRA, Grupo de Investigación en Robótica Autónoma del CAETI: Physical Etoys http://tecnodacta.com.ar/gira/ (last access May 1st, 2013).

Hyun Seung Son, Woo Yeol Kim; Kim, R., Semi-automatic Software Development Based on MDD for Heterogeneous Multi-joint Robots. In Future Generation Communication and Networking Symposia, FGCNS '08. : 2008 , Page(s): 93 – 98 (2008)

Iborra, A.; Caceres, D.; Ortiz, F.; Franco, J.; Palma, P.;Alvarez, B.; Design of Service Robots. Experiences Using Software Engineering. IEEE Robotics & Automation Magazine 1070-9932/09/ IEEE Page(s): 24 – 33. MARCH 2009

Jawawi, D.N.A.; Deris, S.; Mamat, R.; Early-Life Cycle Reuse Approach for Component- Based Software of Autonomous Mobile Robot System. Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. Ninth ACIS International Conference on, Page(s): 263 – 268. (2008)

Jorges, S.; Kubczak, C.; Pageau, F.; Margaria, T.; Model Driven Design of Reliable Robot Control Programs Using the jABC. Engineering of Autonomic and Autonomous Systems, 2007. EASe '07. Fourth IEEE International Workshop on, Page(s): 137-148 (2007)

Jung, E.; Kapoor, C.; Batory, D.; Automatic code generation for actuator interfacing from a declarative specification Intelligent Robots and Systems, 2005. (IROS 2005). IEEE/RSJ International Conference on. Page(s): 2839 - 2844 (2005)

Microsoft, "Microsoft robotics developer studio," 2009,http://msdn.microsoft.com/en-us/robotics/default.aspx, visited on March 11th 2009.

Min Yang Jung; Deguet, A.; Kazanzides, P.; A component-based architecture for flexible integration of robotic systems Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, Page(s): 6107 - 6112 (2010)

Nesnas, I., Wright, A., Bajracharya, M., Simmons, R., Estlin, T.: CLARAty and challenges of developing interoperable robotic software. In: Procs of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), vol. 3, pp. 2428–2435 (2003)

OMG Robotics Domain Task Force (Robotics DTF) http://robotics.omg.org/ (access Nov 1st, 2013).

Pons, C., Giandini, R., Pérez, G., "Desarrollo de Software Dirigido por Modelos. Teorías, Metodologías y Herramientas", Ed: McGraw-Hill Education. ISBN: 978-950-34-0630-4 (2010)

Pons, C., Giandini, R., Arévalo, G., A systematic review of applying modern software engineering techniques to developing robotic systems. Revista Ingeniería e Investigación Vol. 32 No. 1 (2012)

Poppa, F.,Zimmer, U.: RobotUI - A software architecture for modular robotics user interface frameworks, 25th

IEEE/RSJ International Conference on Robotics and Intelligent Systems, IROS 2012; Vilamoura, Algarve; Portugal; Pages 2571-2576; (October 2012).

Sanchez, P; Alonso, D; Rosique, F; Alvarez, B; Pastor, J; Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications. Computers, IEEE Transactions on, Issue: 99 (2010)

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, book edited by Ashish Dutta, ISBN 978-953-307-941-7. (2012)

SmartSoft. [Online]. Available: http://smart-robotics.sourceforge.net/(2013, Jul.)

Stahl, M Voelter. Model Driven Software Development. John Wiley. (2006)

Steven, K., Juha-Pekka, T., Domain-Specific Modeling. John Wiley &Sons, Inc. 2008. (2008)

Szyperski, C., Component Software: Beyond Object-Oriented Programming. 2nd ed. Addison-Wesley Professional, Boston ISBN 0-201-74572-0 (2002).

Thomas, U., Hirzinger, G., Rumpe, B., Schulze, C., Wortmann, A: A new skill based robot programming language using UML/P Statecharts. Proceedings - IEEE International Conference on Robotics and Automation, ICRA 2013; Germany; Pages 461-466; (May 2013).

Tsai, W.T., Qian Huang, Xin Sun. A Collaborative Service-Oriented Simulation Framework with Microsoft Robotic Studio® Simulation Symposium, ANSS 2008. 41st Annual, Page(s): 263 – 270 (2008)

Wei Hongxing; Duan Xinming; Li Shiyi; Tong Guofeng; Wang T.; A component based design framework for robot software architecture. Intelligent Robots and Systems, IEEE/RSJ International Conference on, Page(s): 3429 - 3434 (2009).

Yang, T.-H., Lee, W.-P.: A service-oriented framework for the development of home robots. International Journal of Advanced Robotic Systems; Volume 10, Article number 122; ISSN: 17298806; (February 2013).