Available online at www.sciencedirect.com

ScienceDirect

Procedia Technology 15 (2014) 605 – 612

Procedia Technology

2nd International Conference on System-Integrated Intelligence: Challenges for Product and Production Engineering

Model driven engineering for the implementation of user roles in industrial service robot applications

Alexander Bubeck*, Benjamin Maidel, Felipe Garcia Lopez

Fraunhofer IPA, Nobelstr. 12, Stuttgart, Germany

Abstract

The implementation and installation of today's service robot applications into industrial processes is a challenging and time taking task that is usually executed by domain experts. The aim of the PRACE EU-project is to bring a mobile dual arm service robot into small part assembly tasks, which should be used by regular workers. To fulfill this goal a model driven engineering (MDE) tool chain was developed that separates three different user roles in the development process of an industrial service robot application. For each of these roles, the spectrum of required knowledge is reduced, especially for the role of the end user. Based on the implementation of this separation, an evaluation during the development of the PRACE demonstration use case was done.

Peer-review under responsibility of the Organizing Committee of SysInt 2014.

Keywords: Service robotics; tool chain; model driven engineering

1. Introduction

The implementation and installation of today's service robot applications into industrial processes is a challenging and time taking task that is usually executed by domain experts. Such systems, as e.g. the service robot platform rob@work 3 [1], are complex, software-intensive, and highly integrated systems with impressive capabilities for mobile manipulation in unconstrained environments. However, the thereby introduced high costs and

* Corresponding author. Tel.: +49-711-970-1314; Fax: +49-711-970-1008.
E-mail address: alexander.bubeck@ipa.fraunhofer.de

Peer-review under responsibility of the Organizing Committee of SysInt 2014.
doi:10.1016/j.protcy.2014.09.021

small flexibility, due to the lack of end user development tool chains, result in very few actual installations in the field.

In the last years multiple of functionality-rich robot software frameworks were developed to improve reuse during the development process and to lower the entry barrier for new developers. Most notably, Orocos [2], OpenRTM [3], Player [4] and ROS [5] currently call the attention of the robotic community. These frameworks extend the general ideas of component-based software design and development through reusable software components with well-defined interfaces as shown in [6]. Most of these frameworks promote the development of capabilities in the whole robotics community making a large corpus of reusable functionality available. However, application development using these frameworks still requires a broad spectrum of system knowledge.

The aim of the PRACE EU-project (http://prace-fp7.eu) is to bring a mobile dual arm service robot into small part assembly tasks and regular workers should be able to use the system eventually. Therefore, the improvement of the usability for this target group is one of the development efforts within the project. The solution, shown in this work, uses a model driven engineering (MDE) approach to separate the end user from the other user roles during the development process of robotic applications. In contrast to other applications of MDE in robotic domains, as e.g. in Smartsoft [7] or Proteus [8], this concept is not mapping unified modeling language (UML) profiles to robotic software, but is creating a domain specific language (DSL) directly for the existing robot framework ROS. Hence, current development workflows of the robot domain experts can be directly integrated in to the MDE workflow, thus, fast adoption of the tool chain is supported. Furthermore, proprietary ROS components, which are not created based on the MDE tool chain, are usable and transparent to the end user.

Section 2 of this work presents the MDE approach for the separation of user roles. In Section 3 and 4, we describe the actual implementation of the approach in the BRIDE tool chain and the usage in the PRACE scenarios.

2. Approach

2.1. Model driven engineering

The aim of model driven engineering is to encapsulate complexity and to enforce interfaces, architectures and user roles for a specific software domain. Therefore, it gives the chance to increase the quality of software, as many applications in computer science have shown. The Object Management Group, which is a worldwide organization for model driven software approaches, defines model driven engineering in a multiple layer architecture that form a MDE tool chain. The lowest layer (M0) is the actual running code that conforms to a specific implementation model (M1). The mechanisms to describe this implementation model are defined in the meta-model (M2) layer. Usually, an additional layer gives the tool chain the mechanism to describe meta-models. This layer is the meta-meta-model (M3). This model driven engineering concept is also visualized in the overview Fig 1. The application of this MDE architecture to the ROS component framework makes it possible to model the different concerns in a meta-model description. Based on this, the different end-users can implement specific models for the concerns of their development phase. MDE then gives the opportunity to auto-generate implementation code based on the models specified in the M1 layer. As the framework dependent aspects of the component are explicitly modelled in the M1 layer, the code generation can separate the code into framework-dependent and framework-independent parts. When the code generation is developed in that way, the capability can easily be reused in other robot development frameworks.

2.2. Development processes for robotic applications

Creating a whole robot application requires the developers to transition through a number of different process steps (as e.g. in ISO/IEC 12207). Each of these steps has a different focus and therefore requires different knowledge by the developer. That is why a separation of the user roles during the overall development process should be implemented. By analysing the overall development process on the service robot platform rob@work 3 [1], the knowledge required by the developer was divided into the following roles with their corresponding tasks:

- *Capability building role:* The developer has to implement computational functionality of the component (e.g. a vision algorithm). Additionally, he has to provide the communication mechanisms of the component.

- *System deployment role*: The developer has to configure the capability components and compose them. Coordinators for subsystems might be implemented, too.

- *Application building role*: The developer implements the coordination and adapts some configuration of the system. The end user role can be separated, allowing the developers to only cover the domain they are experts in.

When mapping these roles on a domain specific language (DSL), a separation can be done regarding the artifacts in the model. Based on this separation, different models and model views can be presented to the corresponding user and a separation of user roles can be implemented (see Fig. 1).

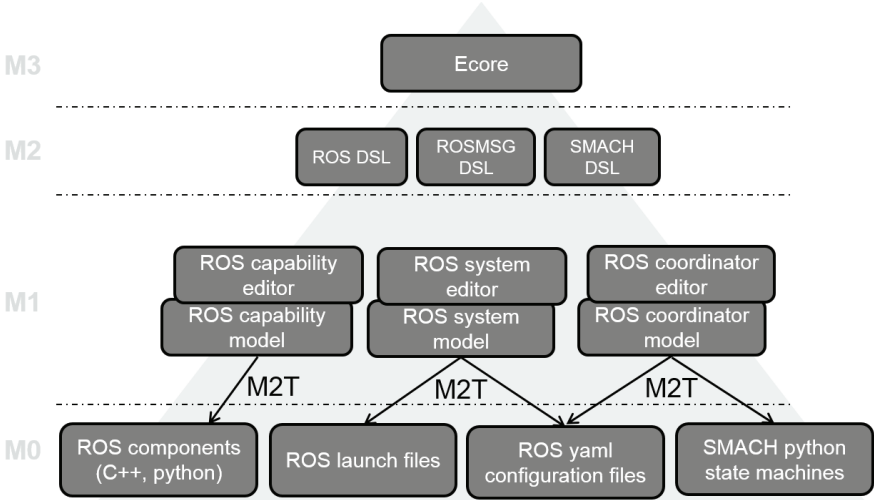

Fig. 1. Overview of model driven engineering approach applied to ROS, with different MDE layers (M0-M3), the integrated concepts of separation and the manifestation in the Eclipse based tool chain.

3. Implementation of a MDE tool chain for the separation of user roles in ROS

The Eclipse Modeling Framework (EMF) and associated projects provide a toolbox to develop model driven tool chains. In contrast to the UML-based SmartSoft [7] tool chain, the work presented here is solely based on EMF. More precisely, the EMF language Ecore (M3) is used to define a domain specific language (DSL) for ROS on the M2-layer. Based on this DSL, an infrastructure including a graphical editor and code generator was developed. This infrastructure, called BRIDE, allows to model capabilities, coordinators and systems in an explicit manner with respect to the corresponding phase in the development process. The code generator creates runnable ROS code and structures the software into platform-independent parts. In the following sections, more details of this implementation will be shown.

3.1. Creation of Meta Models

Based on the general programming abstractions and concepts available in ROS [5], we developed multiple meta-models on the M2 layer representing the tool chain artifacts. The model is the origin of our visual domain specific language and the code generation facilities. The concepts, which are modelled in the meta-model, are available as primitives in our tool chain for later use on the M1 layer. To ease model iteration and tooling, the model is structured as a tree with the architecture primitive as root node. Furthermore, the package contains topics, services, nodes and actions as core primitives (see [5]). The meta-model is separated in five different concerns, namely Computation, Coordination, Configuration, Communication and Composition. This separation enables the systematic formalization of constraints for each concern. For instance, a Service demands at least one Service Server (Communication) and every Parameter demands a name, type and value (Configuration). These and more constraints are checked in the code generation facilities.

3.2. Graphical Interfaces

Based on the defined ROS M2 model, which represents the different aspects of a component, graphical editors for creating component models (M1) can be derived. The generation of multiple editors based on one domain specific language was chosen to map the different aspects formulated in the DSL to the user roles in the development process. Because the editors are developed using a model based process in Eclipse, they are directly linked to the ROS Ecore model. Changes in the Ecore model are therefore directly reflected to the graphical editors. The graphical editor for the first user role, the *capability building role*, consists of tools for defining nodes, services, publishers/subscribers, actions and parameters. Therefore, the capability user is not able to compose different components, but only defines aspects of a component that act on a single component. The created components are organized in a ROS package as defined in the ROS M2 model. The system deployer graphical editor (shown in Fig. 2) implements a tool set for the composition of the capabilities. The user is able to connect components with topics, services and actions and can add packages, including their respective nodes, to the system. Furthermore, parameters defined by the capability building user can be modified. As the capability models are linked to the system model, changes in the capabilities are directly represented in the system editor. Both M1 models, capability and system models, that are created using the graphical editors, are also manifested in a textual XML file that is directly synchronized with the graphical editors making the usage of non-graphical editors for the users possible. A graphical editor for a coordinator model, which can be used by the application developer, is created in a similar fashion.

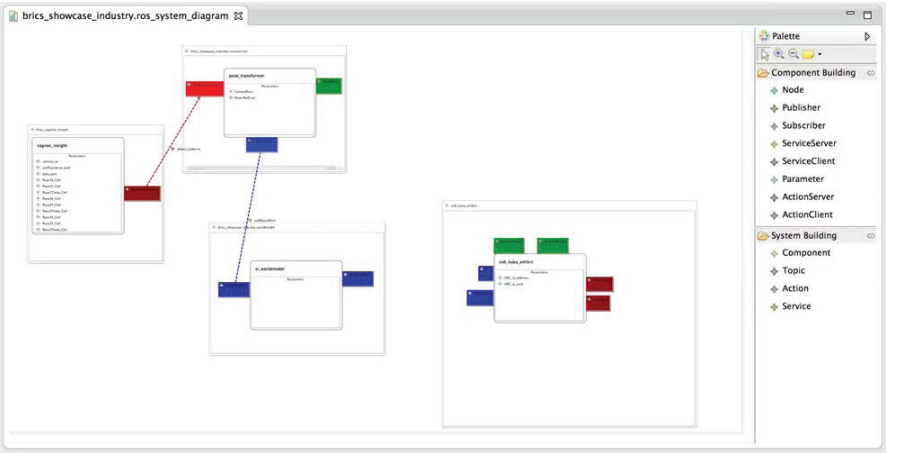

Fig. 2. Screenshot of the system editor of BRIDE.

3.3. Code Generation

Once the capability, system and coordinator models are created, they are automatically transformed to another representation, usually to source code. This so-called model-to-code generation was realized by implementing templates for the Epsilon generator, which is part of the Eclipse tool chain. This approach makes it possible to enforce code quality and standards. It also transparently supports different target languages for the implementation. In case of the capability model, templates for the generation of C++ ROS components and Python ROS components were realized. The code generation takes care of creating a node skeleton as well as the necessary tool chain files such as manifest.xml, the dynamic reconfigure files, the CMakeLists.txt etc., finishing with a ready-to-compile C++ or Python ROS component. In the prepared user code part of the package, the capability builder can now implement the specific capability independent of the component around it. The information described in the system deployment model is transferred to a ROS launch file starting the nodes, renaming the publisher and subscriber topics to the configured ones and setting the parameters to the configured values. The auto-generated code and the implementations of the user can be distinguished clearly. Hence, the model-to-code generation can be executed multiple times, e.g. when there are changes in the model by the user. Due to this fact, the templates for the different targets can be enhanced iteratively and the improved code can be easily disseminated to many implementations.

4. Using BRIDE for the development of a PRACE scenario

4.1. Pick and place scenario

One of the PRACE use cases is a needle rearrangement scenario. Arranged batches of small metal needles must be rearranged to different pallets for different material treatments. Fig. 3 shows a typical needle rearrangement scenario. On the left side is the source pallet where the needles should be picked from and on the right side is the target pallet where the needles should be placed in.

Fig. 3. Needle rearrangement scenario.

4.2. Robot System

The PRACE demonstrator consists of the mobile rob@work 3 [1] platform developed by Fraunhofer IPA, ABBs dual arm concept robot and two pneumatic gripper systems developed by Bosch. On top of the robot, there is also a pan-tilt unit with a camera system mounted (see Fig. 4). The complexity of this system is quite high. It has an overall of 26 degrees of freedom, three computer systems, two Sick laser scanners, a stereo vision camera system and a separate controller for the dual arm manipulator.

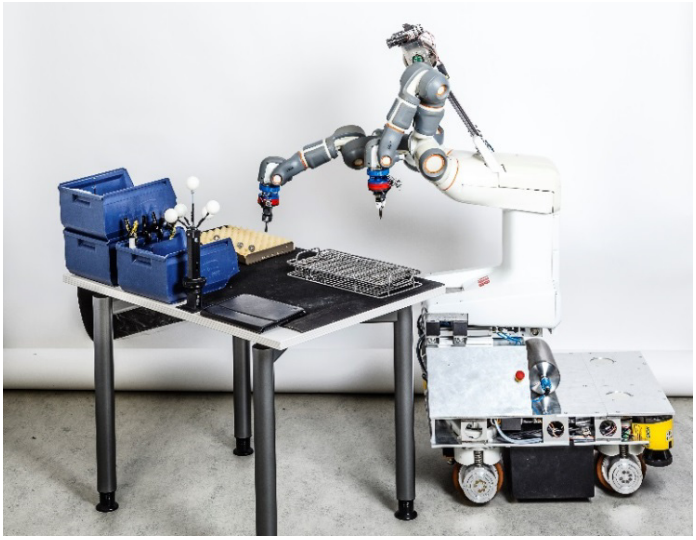

Fig. 4. PRACE demonstrator.

4.3. Application development with BRIDE

During the implementation of this use case, the BRIDE tool chain and the concept of different user roles were applied as described in the previous chapters.

In the first role (*capability building role*), the projects robot developer experts implemented the necessary robot capabilities. Components for path-planning, trajectory execution, gripper controllers and palletizing logic were implemented by the corresponding domain experts. All these components are offering a common interaction interface based on actionlib [5]. Those are specified in the component models created by the BRIDE capability developer editor. For example, the gripper controller offers the functionality ‘open/close gripper’, the path-planning module expects artesian goal positions or goal positions in the joint space. During the project, the domain experts were developing and improving their components with minimal interference with each other.

In the second role (*system deployment role*), which was executed mostly during the integration workshops of the project, the different capability components were configured and composed using the BRIDE system-modelling tool.

In the last role (*application building role*) the end-user created the actual pick and place application by using BRIDEs coordinator model. Inside this model, state machines with different types of states can be added and logically connected by transitions. Each of these states corresponds to one of the capabilities developed in the first phase. An example of a simple pick and place coordinator is shown in Fig. 5. The first state ‘MoveArmToSource’ connects to the path-planning module. The configuration of this state is the position where the manipulator should move to. The second state connects to the gripper module and the configuration is the command to close the gripper. The next two states are exactly the same as before except a different configuration. The transitions are triggered based on standardized outcomes of the different states.

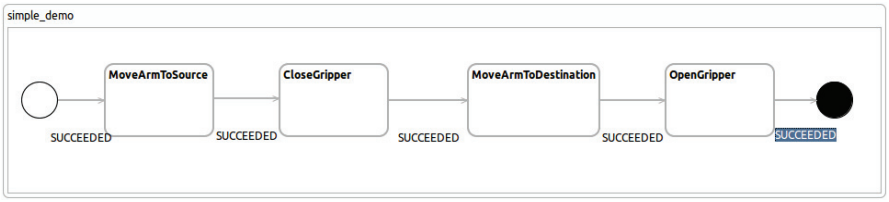

Fig. 5. BRIDE simple pick and place coordinator.

The coordinator model which solves the above described needle rearrangement scenario is shown in Fig. 6. This scenario originates from the simple scenario shown in Fig. 5 and was developed in several iteration-cycles by the end user during the project. In addition, the components developed by the robot domain experts have been improved iteratively during the same time. This shows, there are little inferences between the actual application development and the components development.

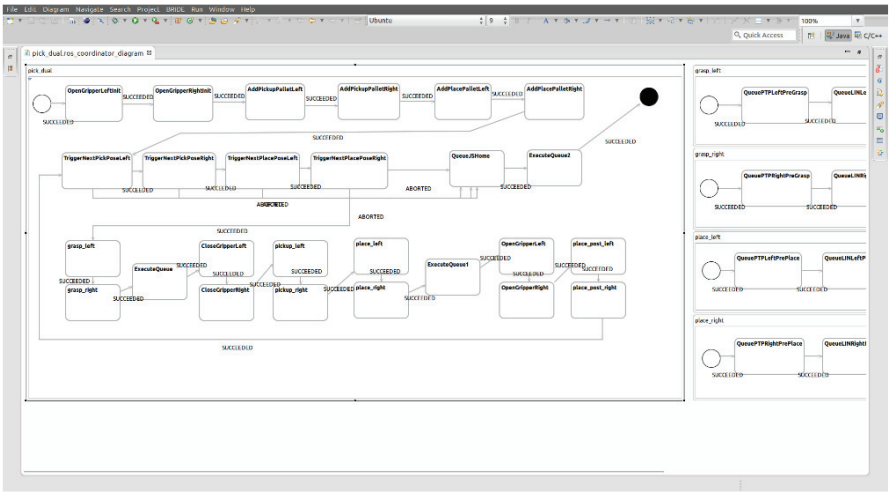

Fig. 6. BRIDE needle rearrangement scenario.

5. Conclusion

Adopting the concepts of model driven engineering to robotics has high impact on reusability of existing software and the performance of robot application, system and capability development, as shown in this work with the application on the PRACE needle picking use case. Additionally, the tool chain BRIDE is actively used and promoted in the ROS community, e.g. as part of the ROS industrial initiative (http://rosindustrial.org/). Because of the growing acceptance in the community, the tool chain is continuously extended to more robot frameworks (e.g. OROCOS) and robot applications (e.g. for AGV systems).

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 FoF.NMP.2011-2 - under grant agreement No 285380 - PRACE.

References

[1] rob@work 3, http://www.rob-at-work.de
[2] Bruyninckx H, Soetens P, Koninckx B. The real-time motion control core of the Orocos project. Proceedings of the IEEE International Conference on Robotics and Automation; 2003. p. 2766-2771
[3] Ando N, Suehiro T, Kotoku T. A Software Platform for Component Based RT-System Development: OpenRTM-Aist. Proceedings of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots, ser. SIMPAR '08, Springer-Verlag; 2008. p. 87-98
[4] Gerkey B, Vaughan R., Howard A. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. 11th International Conference on Advanced Robotics (ICAR 2003), Coimbra, Portugal; 2003.
[5] Quigley M, Conley K, Gerkey BP, Faust J, Foote T, Leibs J, Wheeler R, Ng AY. ROS: an open-source Robot Operating System. Proceedings of the Workshop on Open Source Software held at the International Conference on Robotics and Automation (ICRA); 2009.
[6] Brugali D, Shakhimardanov A. Component-based Robotic Engineering Part II: Systems and Models. IEEE Robotics and Automation Magazine 2010;17(1):100-112.
[7] Schlegel C, Steck A, Brugali D, Knoll A. Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering. Simulation, Modeling, and Programming for Autonomous Robots - Second International Conference, SIMPAR 2010, Darmstadt, Germany. Proceedings, vol. 6472. Springer; 2010. p. 324-335
[8] Lortal G, Dhouib S, Gerard S. Integrating ontological domain knowledge into a robotic DSL. Proceedings of the 2010 international conference on Models in software engineering MODELS'10. Berlin, Heidelberg: Springer-Verlag; 2011. p. 401-414