Best Practice in Robotics (BRICS)

Grant Agreement Number: 231940

01.03.2009 - 28.02.2013

Instrument: Collaborative Project (IP)

Deliverable D4.1: First established CAE tool integration

Peter Soetens, Hugo Garcia, Markus Klotzbuecher, Herman Bruyninckx

Deliverable: D4.1

Lead contractor for this deliverable:	Katholieke Universiteit Leuven
Due date of deliverable:	March 1, 2010
Actual submission date:	April 9, 2010
Dissemination level:	Restricted/Public
Revision:	0.5

Executive Summary

A Model Driven Engineering (MDE) tool chain serves to shape the work flow of an engineer working with the BRICS hardware, software and algorithms. It allows to integrate these elements by using models that describe them and provides an interface for integration. By definition, a tool chain integrates tools into a work flow and this is the main purpose of the BRICS IDE (BRIDE).

In order to facilitate this integration a common language is required. Since BRIDE works with models, the consortium chose to define the BRICS Component Model (BCM) as the mediator between all tools. Each legacy tool that needs integration into BRIDE needs to understand in whole or in part the BCM and to provide a compliant interface, or the BRIDE developers team has to make a *BCM wrapper* around the provided interfaces of the legacy tool. The latter approach is more likely, since few or even no legacy tools will do the effort to make themselves "BCM compliant".

Finally, engineers will use BRIDE to model new components using the BCM and generate running systems from these models. Because of its central role, it is of great importance that the BCM is accepted by the whole consortium and in the future by robotics engineers world wide.

This deliverable describes the road so far into creating BRIDE. Much effort has been spent in creating the BCM and the process on how to define, extend and maintain it. That process is motivated by pragmatism: it starts with the smallest (conceptual) "common denominator" of all existing robot software frameworks; it requires (for the time being, textual) tool chain support for each of its features; and it will only grow when motivated and well-documented use cases are provided by the consortium. A proof of concept model-to-code generator using the BCM has been setup which creates two communicating components. We also started putting the first BRICS user interface elements into the Eclipse IDE, which will serve as the platform for BRIDE.

Contents

1 Contributions to Project Objectives

The work described in this document contributes to the general objectives of the project, as described in the *Description of Work* (DoW):

> *The main objectives of BRICS are accordingly*
> - *to significantly promote the interoperability of hardware and software components by harmonizing the interfaces and as well as the communication and data exchange between these components;*
> - *to design and implement an integrated development environment for robotics (called "BRIDE") and an accompanying software repository of best practise robotics algorithms (called "BROCRE").*

The contributions to these objectives are as follows:

- The *task force* behind BRICS Component model (BCM), Chap. 2, has made a very in-depth study of existing robot software frameworks (Orocos [1], OpenRTM, [3] ROS [7],...) and identified how their designs match with the BCM and/or how bridges between those frameworks and a BCM-compliant software system must be made.
- The first steps towards BRIDE and BROCRE have been made; the progress is estimated to be about 15% of the total required effort towards this goal.
- A significant amount of the efforts by participants to this Deliverable have contributed to the various aspects and needs of *harmonization* of complex robotics software systems, that is, to the facilitation of the *interoperability* between software components provided by different “vendors”, as well as to the *reusability* of all such software components in various application contexts. The results of these efforts are described in the Deliverable of Work Package 8, on *Harmonisation*: *D8.1 Design principles, implementation guidelines, evaluation criteria for harmonisation and benchmarking and use case implementation, foreseen for Month 18.*

2 BRICS Component model (BCM)

The first year into the project, much effort has been spent in creating the BRICS Component Model (BCM), and the process on how to define, extend and maintain it. That process is motivated by pragmatism: it starts with the smallest (conceptual) "common denominator" of all investigated robot software frameworks of significant size and penetration, Orocos [1], OpenRTM [3], ROS [7], OPRoS[1]. The process requires a (for the time being, textual) tool chain support for each of its features; and it will only grow when motivated and well-documented use cases are provided by the consortium.

The BCM serves to define the interface on how components can be integrated into the BRICS MDE tool chain. It also exposes to legacy CAE tools how BRICS models components and facilitates bridging towards such tools. The BCM is the means on how the MDE Toolchain structures component based applications, and how it allows to integrate the deliverables of all work packages. For example, Work Packages 2 *Middleware* and 8 *Harmonisation* are closely related, and the BCM allows to integrate the results of these packages.

The technical description and motivation of the BCM is attached as Appendix A: "BCM: A Minimal Robotic Component Model for Multitarget System and Component Generation". (This Appendix contains more technical details, and is expected to evolve into a scientific publication; it currently still has a *draft* status.)

2.1 Component model efforts

A task-force with core members from K.U.Leuven, BRSU and UBergamo was formed to accelerate the definition of the BCM. The on-going efforts of the task-force have yielded the above-mentioned pragmatic process, and the results that are further described below and in Appendix A.

2.1.1 A formal Meta-Model description

The state-of-the-art Eclipse ECore format was chosen to formally describe the BCM, because it is *the* appropriate and widely applied technology in the Eclipse ecosystem. One speaks of a "Meta-Model" since it describes the primitives available to model components. The inspiration came from both academic references and robotics component models in the field. The current Ecore model describes BRICS components as having flow ports, operations and properties. A system can then be assembled by connecting components to each other. The Ecore model is detailed in Appendix A.

Choosing this world-wide accepted standard to write down the BCM has many advantages. The tooling for creating the BCM comes for free, tools on different levels and from different vendors understand this format. Also maintaining it is not the burden of the members of this project and as such guaranteed for an extended period after the project.

The task-force uses the Eclipse MDE environment to create models and save them in the ECore format. A tool called 'rgen' is used to generate code from these models.

2.1.2 Meta model considerations

This approach of starting from a minimal model conflicts with the more theoretic approach to build the ideal component model up front. As stated above the first approach was chosen for several pragmatic reasons. Firstly, the goal of the workpackage is to create a working toolchain which can support robotic engineers in practice and not to define a perfect component model.

[1] No English-language information available yet...

Secondly, features used in the modeling process must largely exist already on the target platform and can only rarely be added during the model transformation process. Thus, as a rule of thumb features must exist on one or more target platforms in order to be supported in the component model. At last, we believe it is hard if not impossible to forsee the ideal component model. The use case process was created in order to derive the model best suited for supporting developers.

2.1.3 A process for BCM definition

The purpose of this process is to limit and motivate the elements of the BCM. Each element present in the BCM must be motivated by a use case. Use cases are purely expressed as user models consisting of one or more components exercising a certain feature of the BCM. For example, one use case may model reading and writing a component's property. This model is fed into a code generator (see 2.3) which generates code for each supported robotics framework. Together with the use case model, a unit test is provided that exercises the generated code.

The task-force will defend and explain the component model by upholding these use cases. As such, this BCM definition process shifts the discussion from a theoretical model to practical use cases. Rejection of use cases may lead to removal of features of the BCM or vice versa. The ultimate use cases to defend the BCM are the show cases of the DoW, which describe complete systems. The task-force is currently working on more simpler use cases to polish the work-flow.

This process allows use cases to be monitored for quality. Each use case is weighted by these criteria:

- *Is it doing its case in the most simple way ?* Use cases can be challenged for their simplicity by an alternative use case that accomplishes the same task with less code or simpler concepts.
- *Is it supported by many robotics software platforms ?* Use cases lead to code generated for different robotics middlewares. A use case working on more than one middleware has stronger support to be best-of-practice than a use case working on only one.
- *Do other use cases depend on it ?* Is the use case a dead end or a building block for more complex, system level use cases ? A use case having many dependees is in a stronger position than one with few or none.

In general, a use case that has no support on any existing robotics platform should not be accepted to introduce a feature in the BCM. As a special exception, a use case may propose an extension that no robotics framework has and serve as the motivation for such an extension. No other use cases should depend on such use cases. The task-force believes that only few use cases will be of this nature.

2.2 Future work on BCM

The task-force is assembling a template for use cases such that any willing contributor to the BCM definition process can create a motivated and complete case. The use cases and template are an integral part of the BCM.

There are still many unresolved cases that need solving such that the BCM can mature. Noteworthy are:

1 What is the interface that the BCM assumes is present to access operating system resources and component primitives such as ports and properties ?
2 How will composition be modelled ?
3 What is the role of containers, such as found in the CORBA 3 Component Model and should these be modeled ?
4 Does system level modelling also include non-software components, similar to what SysML proposes[4] ?

5 How are legacy components integrated in a system model ? Is their interface discovered at run-time (introspection) or are special model elements necessary ?
6 How to specify the programming language of a component and how to define interoperability between components written in different languages ?
7 How to support communication between components running in different robotic frameworks in heterogeneous systems ?

The task-force hopes to formulate the answers to all these questions by proposing new use cases.

2.3 BRICS component compiler

The BRICS component compiler is a command line tool implemented using the RGen modelling framework [6]. RGen is a lightweight implementation of the Eclipse EMF framework which allows very easy creation of meta-models, model to model and model to text transformations. Compatibility with Eclipse is guaranteed as the Eclipse Ecore meta-meta model is used for defining meta-models. By using RGen the BRICS component model and associated model transformation could be validated very efficiently without the need of Eclipse.

We are currently investigating if we will continue using RGen for the lower level transformation and thereby separating it from the graphical Eclipse based toolchain. The advantage of this is that the BRICS component compiler can be used without the Eclipse dependency therefore allowing component transformation easier to be integrated into build infrastructure such as Makefiles and secondly allowing power users to bypass the graphical toolchain.

The tool currently supports generation of ROS [7] and Orocos/RTT [1] components using DataFlow primitives. The necessary build infrastructure such as Makefiles etc. is automatically generated so that the resulting components can be compiled and run on the fly.

3 Eclipse based MDE toolchain and strategy for legacy tool integration

This chapter describes the development of a Model Driven Engineering (MDE) tool for robotics based on the Eclipse Platform.

3.1 The Goals

We want to develop a specific tool and environment for building components in robotics. To develop tooling that will support a best practice methodology for developing component base software. To implement a practical approach in developing and evolving the tooling. To use standard publicly available software with a high bias to the tools available from the Eclipse Foundation. To support a baseline standard robotic system. To provide a useful product for the robotics community.

3.2 Structure of the Initial version of BRIDE

The BRIDE distribution will be based on plug-ins based primarily from the projects found in the Eclipse [2] Foundation. The initial projects included in the first release of BRIDE are:

- Eclipse Project
  - Java Development Tools (JDT)
  - Plug-in Development Environment (PDE)
  - Eclipse Platform
- Eclipse Modeling Project
  - Graphical Modeling Framework (GMF)
  - Model Development Tools
    - Object Constraint Language (OCL)
  - Eclipse Modeling Framework Technology (EMFT)
    - Ecore Tools
  - Eclipse Modeling Framework (EMF)
    - EMF (Core)
    - Model Transaction
    - Validation Framework
- Tools Project
  - Eclipse C/C++ Development Tooling (CDT)
  - Graphical Editor Framework (GMF)
- EGit plug-in used for the Git version control system.

Third party plug-ins that might be included are:

- Subclipse Subversion Plug-in

3.3 Community Building and Involvement Efforts

A repository of projects that directly contribute to BRIDE will be created. This repository will be independent of the other source repositories of the BRICS project. Contribution to the repository will be managed using the Eclipse Development Process or a version thereof. A BRIDE distribution(s) will be available from the repository using an build system equivalent to the Eclipse System. We follow the “Eclipse Way” since it is the goal of the BRIDE team to graduate the project to at least the incubator level in the Eclipse Foundation where the continuity of BRIDE and further contributions from not only the BRICS community but other robotic communities can be well managed and fomented. We intend to contact Ed Merks (lead the top-level

Eclipse Modeling Project as well as the Eclipse Modeling Framework subproject) in order to invite him in the capacity of advisor for the project. Having Ed Merks, or someone at his level, as part of the project is imperative in order to successfully host an Eclipse Day for Robotics (2011) and to graduate BRIDE into the Eclipse ecosystem as per the rules defined by the Eclipse Foundation. At a more local level, we intend to expose BRIDE to the various European user group with the hopes that eventually this user group will serve as one of the incubator developer bases. Since the resource for BRIDE are very limited, we will also try to start semester long internships at KUL for Bachelor level students of Computer Science in order to help with the workload.

3.4 Functional Aspects and Development of BRIDE as an IDE

We have established a minimal work flow for developing robotic component software. The work flow will serve as basis for establishing a structured best practice in creating component software. We will examine other software development environments for robotics from industry and academia in order to analyze their basic work flow and enhance or complement our work flow. One main current focus is OpenRTM. Based on the work flow provided by the the Eclipse IDE as demonstrated above we tentatively propose the following basic work flow:

1 Create a project.
2 Create an empty component.
3 Write a Unit Test for the component functionality.
4 Add and edit the functional code to the component.
5 Compile.
6 Run the unit tests.
7 Deploy components in a simulation environment.
8 Run systems level tests in the simulation.
9 Deploy on hardware.
10 Run systems level test on hardware.
11 Execute and verify functionality.

The above work flow is influenced by currently accepted best practices and principles of Agile Software Development. It is our intention to integrate more of the Agile Process into our our work flow. One current research problem is how to test component software? In answer, we will commence an effort to establish a minimal testing framework based on the combination of unit testing and systems testing.

3.5 Model Driven Engineering Aspects of BRIDE

There are currently two available approaches to providing a MDE abstraction to the work flow in BRIDE as exemplify in Figure 3.1. One flow involves a two phase approach where a Component Builder view representing a base line test system is used to graphically compose Components from its parts (e.g. Ports, Interfaces, Services, etc.). Once a component is defined then we zoom out or switch to a Systems View where the Component is represented as one entity and can be used as one of many assembly blocks for a System. The other approach is to have an integrated Builder/System View where the Component parts are NOT represented as separate graphical entities and instead of represented graphically as internal glyphs to the component (e.g. as methods or fields in standard UML class notation). Both approaches will be implemented an field tested in order to determine which one approach is most appropriate or if there is yet an unforeseen best approach that can be delivered within the scope of the first release of BRIDE.

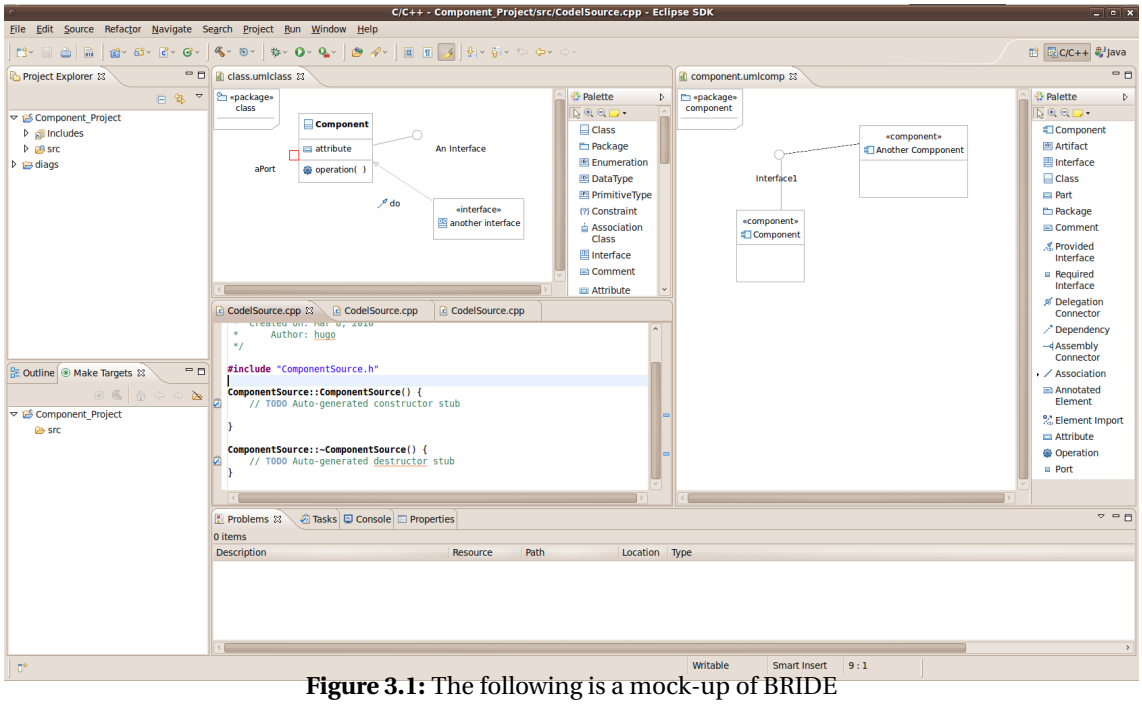

Figure 3.1: The following is a mock-up of BRIDE

3.6 Component Model and BRICS Compiler Integration

Integrating the above MDE approaches and work flow to the Component Model will be a major focus of the work ahead. We have defined a model transformation process as shown in Figure 3.2 which relates the tools with the model transformations. The process is started by creating a user readable and editable model within the Eclipse IDE. The user will be working with an instance of the component model and will at some point have to select or define information specific to the target platform which he intends to use. The instance model is persisted and shared using the standard XMI file format. Next the `bricc` component compiler will be invoked by the toolchain in order to generated platform specific components from instance model and target specific information. After this step the generated code is compiled and the resulting binary component is ready to be deployed. As noted in the figure, the component model serves to validate the instance model and support the `bricc` generation process. Our expectations are that not only the Component Model and the tooling will be validated by the before mentioned processes but that also a synergy will be established where the process is validated and modified by the Component model. We will establish a common ground for interoperability with the BRICS compiler via the use of an XML Metadata Interchange (XMI) representation of the Component System produced with BRIDE. In addition, we will take a practical approach in conjunction with the process mention before and directly produce components from the IDE using one component framework (either Orocos [1] or OpenRTM [3]) using simple templates. This will enable us to rapidly prototype and validate the functionality of BRIDE. It is intended that this interim solution will be dropped once the `bricc` is ready for full BRIDE integration.

3.7 Integrating legacy tools

Each legacy tool that needs integration into BRIDE needs to understand in whole or in part the BCM and to provide a compliant interface, or the BRIDE developers team has to make a *BCM wrapper* around the provided interfaces of the legacy tool. The latter approach is more likely, since few or even no legacy tools will do the effort to make themselves “BCM compliant”.

We have chosen Blender [5] as our first tool to be integrated with BRIDE, because (i) even the simplest version of the BCM can be used to interface Blender, (ii) the programme provides

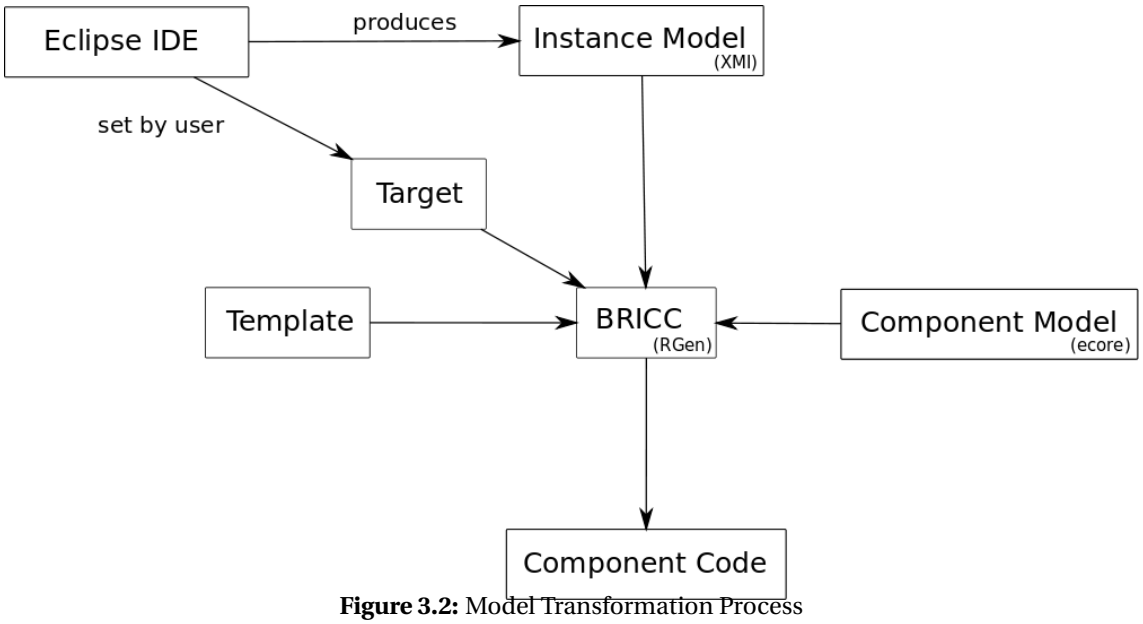

Figure 3.2: Model Transformation Process

attractive 3D visualisation that no other Task in the project provides, and (iii) its *Open Source* nature allows to write a "Blender native" wrapper into the Blender code itself. It is not clear yet how tight the integration can or should be accomplished; e.g., Blender allows loose integration via Python scripts, or via *messages* in its *Game Engine*, but also tight integration via an update of its *C++* code. The simplest solution will be TCP/IP port communication between Eclipse and Blender. Another possibility is the usage of an C or Python (Jython?) API bridge that will allow better communication. At this moment is seems that have a Blender view within BRIDE itself is a technical challenge beyond the resources of the project; anyway, this feature is only a *nice to have* and certainly not a high-priority requirement.

A second, obvious candidate for integration as a legacy tool is the Matlab/Simulink combination of *The Mathworks*, because of the wide-spread usage of this tool, inside the project as well as in the whole robotics community. In addition to loose integration as in Blender, Matlab/Simulink offers another approach, via its code generation functionality: this could be configured to generate BCM-compliant components. The tight integration option is not possible, because of the *closed source* nature of the Matlab/Simulink software.

3.8 Conclusion

We have defined the initial technologies to be integrated into BRIDE. We have provided a first plan for integrating BRIDE to the robotics community and the Eclipse ecosystem. We have established a methodology for the development of BRIDE and the focus of the best practices associated with BRIDE. We have selected a first legacy tool for integrating into BRIDE. It is our opinion that the workload and goals are appropriate within the context of the first concrete deliverable of BRIDE. We will document the process of the building BRIDE using TRAC in order to complete our commitment towards due diligence in our first deliverable.

Bibliography

[1] Herman Bruyninckx. Open Robot COntrol Software. http://www.orocos.org/, 2001. Last visited 2010.

[2] Eclipse Foundation. The Eclipse Integrated Development Environment. http://www.eclipse.org.

[3] National Institute of Advanced Industrial Science and Technology, Intelligent Systems Research Institute. OpenRTM-AIst. ttp://www.openrtm.org.

[4] OMG. SysML: Systems Modelling Language. http://www.sysml.org/.

[5] Ton Roosendaal. Blender. http://www.blender.org. Accessed online 2 August 2009.

[6] Martin Thiede. RGen: Ruby Modelling and Generator Framework. http://ruby-gen.org/.

[7] Willow Garage. Robot Operating System (ROS). http://www.ros.org, 2009. Accessed online 26 Januari 2010.