The BRICS Component Model: A Model-Based Development Paradigm For Complex Robotics Software Systems

Herman Bruyninckx
Katholieke Universiteit Leuven
Leuven, Belgium
Herman.Bruyninckx@mech.kuleuven.be

Nico Hochgeschwender
Bonn-Rhine-Sieg University of Applied Sciences, Germany
nico.hochgeschwender@h-brs.de

Luca Gherardi
University of Bergamo
Bergamo, Italy
luca.gherardi@unibg.it

Markus Klotzbücher
Katholieke Universiteit Leuven
Leuven, Belgium
markus.klotzbuecher@mech.kuleuven.be

Gerhard Kraetzschmar
Bonn-Rhine-Sieg University of Applied Sciences, Germany
gerhard.kraetzschmar@h-brs.de

Davide Brugali
University of Bergamo
Bergamo, Italy
brugali@unibg.it

ABSTRACT

Because robotic systems get more complex all the time, developers around the world have, during the last decade, created component-based software frameworks (Orocos, OpenRTM, ROS, OPRoS, SmartSoft) to support the development and reuse of "large grained" pieces of robotics software. This paper introduces the BRICS Component Model (BCM) to provide robotics developers with a set of guidelines, meta-models and tools for structuring as much as possible the development of, both, individual components and component-based architectures, using one or more of the aforementioned software frameworks at the same time, without introducing any framework- or application-specific details. The BCM is built upon two complementary paradigms: the "5Cs" (separation of concerns between the development aspects of Computation, Communication, Coordination, Configuration and Composition) and the meta-modeling approach from Model-Driven Engineering.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.11 [**Software Architectures**]: Domain-specific architectures

General Terms

Design

Keywords

Software Architectures, Reusable Software, Robotics, Component Models

SAC'13 March 18-22, 2013, Coimbra, Portugal.

1. INTRODUCTION

Software development and integration in robotics is a challenging exercise, which is prone to errors. Today's robotic systems are getting more complex, requiring the integration of motion controllers on many motion degrees of freedom, multiple sensors distributed over the robot's body and embedded in its environment, planners and reasoners for ever more complex tasks. Moreover, the need to integrate functional modules into a complete robot control architecture with quality-of-service demands (e.g. real-time constraints) are factors, which are increasing the overall complexity. To tackle the aforementioned challenges, robotics system integrators and developers created component-based software frameworks [4, 31, 6] (such as Orocos [8, 7], OpenRTM [1], ROS [26], OPRoS [19], GenoM [14, 20], SmartSoft [28], and Proteus [16]) to support software development and to promote reuse of "large grained" pieces of robotics software.

Generally speaking, component-based development complements the more traditional object-oriented programming in roughly the following ways: it focuses on *runtime composition* of software (and not *compile time* linking), it allows multiple programming languages, operating systems and communication middlewares to be used in the same system, and it adds *events* and *data streams* as first-class interaction primitives in addition to method calls. Further, the core idea is to use components as *building blocks* (possibly provided by different "vendors") and to design application architectures by *composing* components.

Even though the component-based approach fosters the structured development of components, the composition of robot control architectures out of components provided by independent suppliers remains cumbersome. Currently, the ad-hoc development typically leads to components that are hard to compose and to reuse as they violate basic software engineering principles such as separation of concerns (e.g. in ROS, a ROS component, also called node, is responsible for retrieving its own configuration from the parameter server, hence the functionality becomes dependent on the

latter mechanism).

Further, even though the mentioned software frameworks are based on the same or similar concepts (see [29] for a survey), most of them do not provide (except for OpenRTM) an explicit and formalized component model as required to make the step from code-driven to model-driven software development in robotics. Such a component model would allow to describe the primitives available within a software framework and the composition rules with which the primitives can be combined into larger components or even complete applications. The model-free approach prevents interoperability between software frameworks. For instance, it is very difficult to reuse a component developed in Orocos in an OpenRTM application because the terminology in both software frameworks obfuscates the similarities. Even the simplest possible data-flow bridge between ROS and Orocos required substantial manual work [30].

In general, the model-driven development paradigm is not very popular in robotics. One possible explanation of the slow adoption of this approach in robotics is that mainstream developers find "modeling" too time-consuming, too abstract, and too long term, in order to be of immediate use. Furthermore, a major hurdle in every model-based initiative is to provide enough added value in an MDE (model-driven engineering) toolchain, such that developers can adopt MDE in their daily workflow.

In this paper we introduce the *BRICS Component Model*[1] (BCM) to provide developers with as much structure in their development process as is possible without going into any application-specific details. This BCM structure is, on the one hand, rich enough to make a real difference and on the other hand, generic enough to be applicable in all of the above-mentioned robotics software frameworks. The BCM itself is a *design paradigm*, in that it introduces a methodology. More precisely, we adopt the OMG's meta-modeling paradigm and introduce platform-independent meta-models with a particular emphasis on separation of concerns. Furthermore, we put much emphasis on the complementary roles of (i) advocating the importance of introducing formal *models* into the robotics development process (instead of the mostly *code-only* software frameworks that are now popular), and (ii) well documented "best practices" that solve particular use cases. The medium-term ambition of the BCM developments is that, both together, will suffice to allow developers to first *model* their components and systems (hopefully starting from a rich model repository with "best practice" sub-systems and components), and then *generate code* from those models, using the best available implementations in available software frameworks.

Overview of the paper. Section 2 reports about related works in the context of component based model and meta model driven engineering. Section 3 explains how meta-modeling can be introduced in robotics, with the envisaged balance between concrete formal structure and simplicity. Section 4 introduces the second important part of BCM, the so-called "*5Cs*" that extend the well know paradigm of the separation of the 4 concerns [27]. In section 5 we discuss a use case which applies the concepts introduced in this paper. Finally, in section 6 we draw the relevant conclusions.

[1]Named after the European research project *BRICS*, Best Practice in Robotics, (Grant number FP7-231940) in which it is developed.

2. RELATED WORK

We discuss related work in the field of component models in robotics and beyond as well as model-driven engineering in general.

2.1 Component Models

When the robotics community became aware that it makes sense to spend time developing reusable software frameworks[2], the *CORBA Component Model* (CCM) was a large source of inspiration. However, the advantages of its component model rather quickly lost the battle against the huge learning curve and massiveness of the CORBA standard: it was considered to be way too heavy and all-encompassing for the needs of the robotics community. So, the CCM was not able of establishing itself in mainstream robotics, except for the (at least in the Western world) less popular OpenRTM, OPRoS, and SmartSoft, and, to a smaller extent, Orocos which has had industrial users since its inception. Nevertheless, the most recent "Lightweight" version of the standard [25] has a focus on realtime and embedded systems, and would fit a lot better to the current robotics needs and mindset in the community. However, ROS has conquered most of the community via its low entry threshold, yet of all the robotics software frameworks mentioned in this paper it is least formalized. The BCM introduced in this paper *shares* parts of its *structural* model with that of the CCM (components, interacting via methods, data streams and/or events, and *composed* by *ports*).

The BCM shares this context and goals with many other engineering domains that require lots of online data processing. For example, some large application domains outside of robotics also have seen the need for component-based software frameworks [11]: *automotive* (with AUTOSAR [9] as primary "component model" standard, and AADL [13] as pioneer modeling language for "computational resources"), *aerospace* (driving UML-based evolutions such as MARTE [23]), *embedded systems* [21, 17], and *service component architectures* in web-based systems [22].

Of the most popular robotics software frameworks in the "Western" robotics community, neither ROS nor Orocos have an *explicit* and *formal* component model, in contrast to OpenRTM ("Japan") and OPRoS ("Korea"), which both use Eclipse [12] as a *programming* tool (but only to a limited extend as a *model-driven engineering* tool). However, also the component model in OpenRTM and OPRoS is semantically poorer and less explicit than the BCM's separation of concerns as explained later.

2.2 Model-Driven Engineering

As major short-term ambition, the BCM wants to introduce into the robotics community the *mindset* that it is worthwhile to provide fully formal models for *all* of the system constituents, as well as an explicit model-driven engineering *development process*, with support of models and development workflows in large-scale toolchain ecosystems such as Eclipse. The motivation for this approach is the success that *model-driven engineering* [2, 3] has seen in domains where *industry* (and not academics) is driving the large-scale software development: the paradigmatic belief is that complex systems can only be developed in a maintain-

[2]This was around 2000, with GenoM [14], Player-Stage [15], Miro [32], Orocos [8] and SmartSoft [28] as pioneers.

able and deterministic way if they are first modeled, analyzed and verified abstractly, and only then the code (in a concrete programming language) is generated. The robotics domain is not that far yet, mostly because of the attitude of software developers that they can produce code faster and better in their favorite programming language than via the "detour" of formal models. This observation is, in practice and in the short term, very often a valid one, since a mature toolchain is necessary to attain the benefits of model-driven engineering. Yet significantly less efforts are put into building such tools compared to, for example, the average programming language compilers[3]. But as soon as the critical developments of (i) model definitions, (ii) model-to-text code generation, and (iii) MDE toolchain support for all phases of the development process (including the currently poorly supported phases of deployment and runtime reconfiguration!), will be realized, the overall development process is expected to speed up with an order of magnitude.

The paradigm of *meta-modeling*, on which our approach is based (as well as the four level pyramid represented in figure 1), has its origin in the *Model Driven Engineering* (MDE) domain of software engineering, aiming primarily at improving the process of *generating code* from abstract models that describe a domain. The MDE terminology for going from a higher to a lower level of specificity or detail in the knowledge of a domain is: from *platform-independent* to *platform-specific*, by adding the *platform knowledge*.

The Object Management Group [24] is the main driver of standardization in this context of Model Driven Engineering, for which it uses the trademarked name *Model Driven Architecture*. A huge software effort is being realized in the Eclipse project ecosystem[4] to support all aspects of MDE.

The four *levels of model abstraction* defined by OMG have, in this paper's context, the following meaning:

M3: the highest level of abstraction, that is, the model that represents all the *constraints*, or *restrictions*, that a model has to satisfy without encoding any *specific* knowledge in a domain. The Eclipse Modeling Framework consortium[5] has standardized the M3 level via its *ECore* meta-meta-model language.

M2: the level of the so-called (in MDE speak) *platform-independent* representation of a domain, introducing models with domain specific names relationships (conforming to the M3-level constraints).

M1: the level of a so-called *platform-specific model*. That is, a concrete *model* of a concrete robotic system, but without using a specific programming language.

M0: the level of an *implementation* of a concrete robotic system, using software frameworks and libraries in particular programming languages.

3. THE BRICS META-MODEL

The general meta-modeling ideas of OMG are applied to the domain of robotics as illustrated in figure 1:

[3]Hence, the BRICS project is working on an Eclipse-based toolchain, BRIDE, [10]. Deeper discussions about this toolchain are beyond the scope of this paper.

[4]`http://www.eclipse.org`

[5]`www.eclipse.org/emf`

M3: The Meta-Meta-Model. On this level we used the *ECore* meta-metamodel provided by the Eclipse Modeling Framework (EMF).

M2: The Meta-Models. On this level we introduced an abstract model and a set of specializations of it. The abstract model is called *Component-Port-Connector* (CPC). This model is universally present (but most often only implicitly) in all robotics software designs, not just for modeling component-based systems but also in many functionality libraries (control systems, Bayesian networks, Bond Graphs, etc.). This meta-model is not unique to robotics but holds in many engineering system contexts where sub-systems interact with each other through well defined interaction points ("ports"). The CPC is specialized by the component models of robotic software frameworks such as ROS and Orocos, which represents the software framework-specific meta-models.

M1: The concrete Models. This level represents concrete models of concrete robotic systems. The important difference with M3 and M2 is that, at M1 level, toolchain support is very important to help developers in the complex tasks of concrete system design or component development. A major design issue to improve "user friendliness" is the ability of such a toolchain to provide the above mentioned models with terminology that is familiar to the robotics developer community.

The BRICS project supports M1 via its BRIDE toolchain, which provides a tool for designing M1 models and a set of Model-to-Model transformations for transforming a model conforming to the CPC meta-model in a model conforming to the Orocos or ROS meta-models. The meta-models and the graphical tool are implemented using EMF and GMF while the M2M and the M2T transformations by means of Epsilon. The following subsections give more details about the core contributions of this paper, namely the abstract level M2.

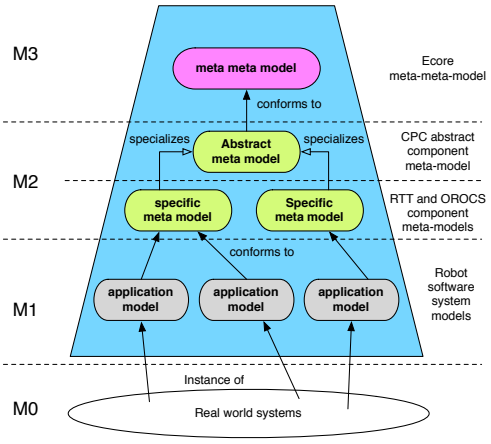

Figure 1: The figure shows how the OMG 4 levels of abstraction are applied in the BCM approach.

3.1 The CPC Meta-Model

This is the simplest, most abstract model introduced in this paper. The Component-Port-Connector meta-model (CPC) is represented in figure 2 and is made of the following parts.

- *Modeling primitives:* A system is a collection of components and connectors. Components provide the *containers* of functionality and behaviour, while the Ports give localized and protected access to the Components' internals. Components can be configured by means of Properties.
- *Composition rules*: Connections represent the interactions between the functionalities and behaviours in two Components, as far as accessible through the Components' Ports.
- *Constraints*: not all compositions make sense, e.g. connecting three Ports to each other directly. Here is a (not yet exhaustive) list of composition constraints. In our meta-model this constraints are expressed as OCL rules.
  - Connections form a *graph*, with Ports *always* in-between Components and Connections.
  - A Component contains zero or more Components.
  - A Component can be contained in only one Component (different from itself).
  - A Component contains zero or more Ports.
  - A Port belongs to one and only one Component.
  - A Connection is always between two Ports within the same composite Component.
  - A Port can be promoted as a composite port, in order to make it accessible to external components. In the same way properties can be promoted as composite properties.

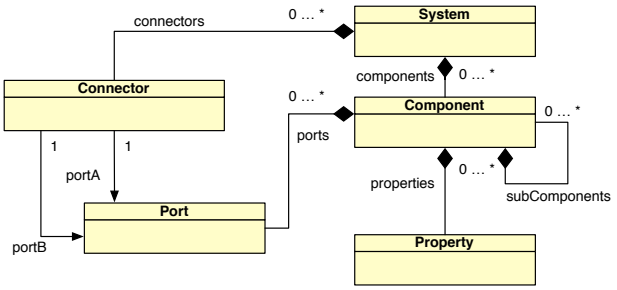

Figure 2: The UML diagram representing the CPC meta-model.

4. SEPARATION OF CONCERNS: THE 5CS

In addition to the model driven approach presented in the previous section, BRICS also promotes the separation of concerns. In order to better apply to the Robotics domain, the well known 4 Concerns (*4Cs*) presented in the seminal work [27], have been extended by separating the original "Configuration" idea into the more fine-grained "Composition" and "Configuration" aspects. Figure 3 gives an overview of how the "5Cs" are defined in the context of the BRICS Component Model (BCM). A more detailed vision of the revised *"5Cs"* is given below.

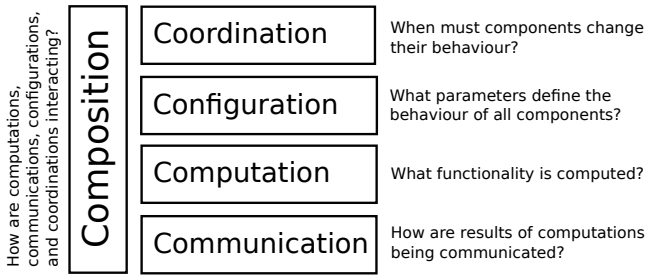

Figure 3: Overview of the 5 concerns.

- **Computation**: this is the core of a system's functionality, and it implements (in hardware and/or software) the domain knowledge that provides the real added value of the overall system. This typically requires "read" and "write" access to data from sources outside of the component, as well as some form of synchronization between the computational activities in multiple components.
- **Communication**: this brings data towards the computational components that require it, with the right quality of service (i.e. time, bandwidth, latency, accuracy, priority, etc).
- **Coordination**: this functionality determines how all components in the system should work together, that is, in each state of the coordinating *finite state machine*, one particular behaviour is configured to be active in each of the components. In other words, Coordination provides the *discrete behaviour* of a component or a system.
- **Configuration**: this functionality allows users of the Computation and Communication functionalities to influence the latters' behaviour and performance, by giving concrete values to the provided configuration parameters (e.g. tuning control or estimation gains, determining Communication channels and their inter-component interaction policies, providing hardware and software resources and taking care of their appropriate allocation, etc).
- **Composition**: while the four "Cs" explained above are all motivated by the desire *to decouple* the design concerns as much as possible (while avoiding to introduce too many separate concerns), the design concern of Composition models the *coupling* that is always required between components in a particular system. Roughly speaking, a good Composition design provides a motivated trade-off between (i) *composability*, and (ii) *compositionality*. The former is the property of individual *components* of being optimally ready *to be reused* under composition; the latter is the property of a *system* to have predictable behaviour as soon as the behaviours of its constituent components are known.

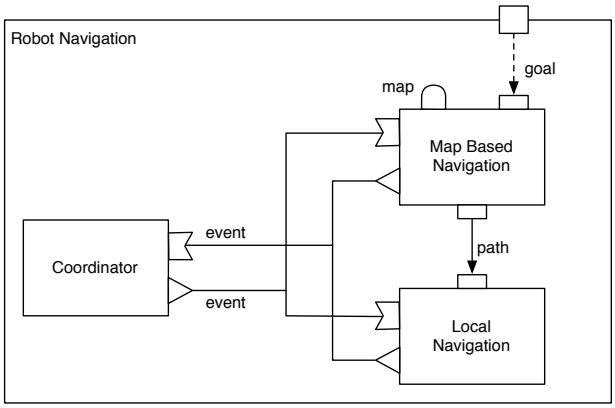

Figure 4: The Robot Navigation composite component modeled with CPC.

5. CASE STUDY

In this section we present a case study that shows how the BRICS Component Model and the separation of concerns have been applied in order to design an architecture for a robot navigation application.

Robot navigation is the ability of a mobile robot to autonomously navigate from its current position towards a goal position, while avoiding dangerous situations such as collisions with obstacles. This ability involves several functionality like the management of the environment representation (i.e. *Map Server*), the computation of the optimal path (i.e. *Path Planning*), the capacity of avoiding moving obstacle and controlling of the physical robot (i.e. *Local Navigation*).

Figure 4 and 5 show how the components of this application are connected and how the architecture is organized as a composition hierarchy. The architecture is modeled using the CPC meta-model and for this reason it is software-framework independent. More in detail the model depicted in figure 4 shows the top level composite component (the *Robot Navigation*), while the model depicted in figure 5 the *Map Based Navigation* composite component, which corresponds to the homonymous component used in figure 4. Components are represented by means of boxes and their ports are depicted with different shapes. Squared connectors represent data flow ports, lollipop-socket connectors represent provided and required services, triangular connectors represent events communication. Finally AND-style shapes represent component properties.

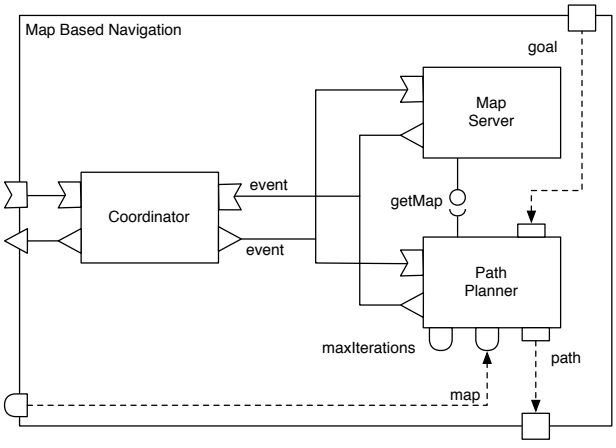

Figure 5: The Map Based Navigation composite component modeled with CPC.

Below we describe how the 5Cs have been decoupled in the design of the depicted components.

Computation. Computational components are concerned with the data processing algorithms required by an application [27]. The components computation is triggered by exchanging information (e.g. algorithm input parameters) by means of data-flow ports and service calls. In the case study the top-level composite component contains two computational components: the *Local Navigation* and the *Map Based Navigation*. The first one receives a path as input and is in charge of driving the robot towards that path by avoiding dynamic obstacles. The second component is itself a composite. It receives as input a geometrical goal and produces as output a geometrical, obstacle-free path. This composite component contains two leaf components: the *Map Server*, which provides the map of the environment by means of a service, and the *Path Planner*, which is in charge of computing the path.

Configuration. Configuration determines how components execute their functionality. These aspects can be configured by setting the values of a set of parameters defined in the components implementation. Parameters can be accessed by means of a mechanism called properties, which can be modified at deployment-time by an apposite infrastructure provided with the software framework. This infrastructure is in charge of deploying the system and configuring the components. In our case study the *Path Planner* component provides two properties: *maxIteration*, which defines the maximum number of iterations allowed for the path planning algorithm, and *map*, which defines which map has to be used.

Coordination. Coordination components are in charge of starting and stopping the computational components of a composite. They also define how computational components work together. Each composite component has to contain one and only one coordination component, which interacts with other components by sending and receiving only events. Coordination components are typically implemented as state machines. [5] and [18] present how this is possible by using respectively the Abstract State Machines and the rFSM. In the case study the upper level coordinator is in charge of coordinating the activities of *Map Based Navigation* and *Local Navigation*. The lower level coordinator instead coordinates the *Map Server* and the *Path Planner*.

Communication. Communication deals with the exchange of data [27]. In the case study we have defined which mechanism we use for each connection between components (e.g. data flow, service call, events). The use of a specific software framework allows the use of different implementations for the same communication mechanism. Furthermore the most common software framework allows us to define the communication policy (e.g. SCA binding, Orocos Connection Policy), which defines for example the lock mechanism or the buffer size.

Composition. Computational and coordination components, including their configuration and communication, can

be hierarchically organized in composites, which can be then reused as components for creating more abstract composites. As well as standard components also composite components provide ports and properties. Their ports and properties belong to internal components and are promoted in order to be part of the composite interface (in the figures ports and properties promotions are depicted by means of dashed lines). External components can communicate with internal components only by means of ports defined in the composite interface. Also in the case study the composition is used for providing the *Robot Navigation* and the *Local Navigation* components.

5.1 From Software Framework Independent to Software Framework Specific

Thanks to the abstraction level offered by CPC, the architecture model presented above doesn't have any software framework specific element. One the one hand this is desirable because it improves the flexibility, on the other hand that model cannot be implemented on a real system. However thanks to a set of model-to-model transformations it is possible to transform a model conforming to CPC in a new model which conforms to the component model of a specific software framework. Once the transformation has been applied the developer is in charge of completing the model by filling it with a set of information that are specific for the desired software framework.

For example the *Robot Navigation* model can be automatically transformed into an Orocos-specific model. This new model will require some additional information about the ports directionality, the connection policies and the components implementations, which can be inserted by the developer by means of the apposite property views provided by our tool (BRIDE).

Figure 6 depicts a screenshot of BRIDE, which supports both the design of M1 models conforming to CPC, Orocos and ROS and the execution of M2M transformations of these models among each other. The depicted model is the result of a M2M transformation (from CPC to Orocos) applied to the model of the *Robot Navigation* described above. Components are represented by boxes, which contain properties and operations. Data-flow ports and event ports are represented by blue and yellow boxes (blue boxes are input ports, while yellow boxes are output ports). Each port reports its name on the top and the data type on the bottom. Events port can be distinguished from data-flow ports thanks to a flag, which is visible in the property view.

6. CONCLUSIONS

The ambition of the BRICS Component Model is to introduce models as *first-class citizens* in the robotics software development process. The major reasons are (i) providing explicit and formal models in a domain helps developers *to structure* their design efforts and outcomes, and (ii) code in concrete programming languages can then be *generated* from the models via a (semi)automatic toolchain ("model-to-text" transformations, in MDE speak), instead of implementing it manually. So, while code generation from models is essential in the ambition of the BCM, its current state of practice is still rather elementary in this context. However, the concepts of the BCM have already been tried on more

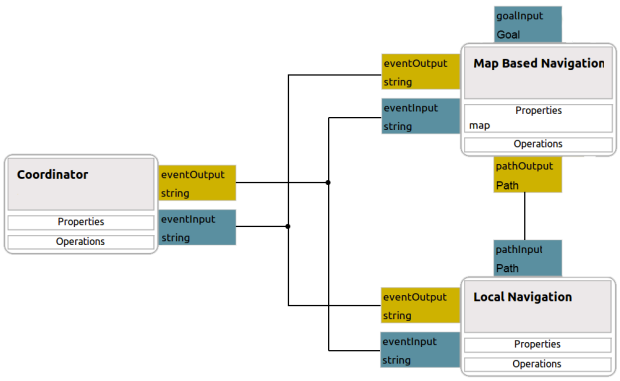

Figure 6: The Map Based Navigation composite component modeled with Orocos.

than one hundred developers (PhD students, and academic and industrial robot software engineers) on various occasions (research camps organized by the BRICS project; or targeted dissemination workshops with selected developers), and the outcome is always positive: these developers quickly get concrete "best practices" from the BCM paradigm, even without full toolchain support or standardization of the "5C" models. However, our ambition of course is that developers adopt the tools developed in BRICS to design and develop future robotic systems such that the impact is measurable.

Currently, the BCM concepts can be applied successfully in the design and development of *new* components in existing "non-BCM-based" software frameworks such as ROS or Orocos, by developers that are sufficiently disciplined to map the BCM and its best practices onto the available component primitives in the software frameworks. However, the other way around will most often fail; that is, it is typically impossible to make a 5C model out of an existing ROS or Orocos system, because those software frameworks are not yet supporting their users to use the 5Cs in a systematic way.

7. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-231940-BRICS (Best Practice in Robotics). The authors would like to thank all the partners of the BRICS project for their valuable comments.

8. ADDITIONAL AUTHORS

Additional authors: Azamat Shakhimardanov, Jan Paulus, Michael Reckhaus (Bonn-Rhein-Sieg University of Applied Sciences), Hugo Garcia (Katholieke Universiteit Leuven), Davide Faconti (Katholieke Universiteit Leuven) and Peter Soetens (Intermodalics).

9. REFERENCES

[1] N. Ando, T. Suehiro, and T. Kotoku. A software platform for component based rt-system development: Openrtm-aist. In *Proceedings of the 1st International Conference on Simulation, Modeling, and*

Programming for Autonomous Robots, SIMPAR '08, pages 87–98, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] C. Atkinson and T. Kühne. Model-driven development: a metamodeling foundation. *IEEE software*, 20(5):36–41, 2003.

[3] J. Bézivin. On the unification power of models. *Software and Systems Modeling*, 4(2):171–188, 2005.

[4] A. Brooks, T. Kaupp, A. Makarenko, A. Orebäck, and S. Williams. Towards component based robotics. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 163–168, 2005.

[5] D. Brugali, L. Gherardi, E. Riccobene, and P. Scandurra. A formal framework for coordinated simulation of heterogeneous service-oriented applications. *8th International Symposium on Formal Aspects of Component Software (FACS)*, 2011.

[6] D. Brugali and A. Shakhimardanov. Component-based robotic engineering part ii: Systems and models. *IEEE Robotics and Automation Magazine*, 17(1):100 – 112, 2010.

[7] H. Bruyninckx. Open RObot COntrol Software. `http://www.orocos.org/`, 2001.

[8] H. Bruyninckx, P. Soetens, and B. Koninckx. The real-time motion control core of the Orocos project. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2003.

[9] A. Consortium. Autosar—AUTomotive Open System ARchitecture. `http://www.automationml.org`, 2003.

[10] B. Consortium. BRIDE—the BRIcs Development Environment. `http://www.best-of-robotics.org/bride/`, 2012.

[11] I. Crnković, S. Sentilles, A. Vulgarakis, and M. R. V. Chaudron. A classifcation framework for software component models. *IEEE Trans. Software Engineering*, 37(5):593–615, 2011.

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

[13] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The architecture analysis & design language (AADL): An introduction. Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, 2006.

[14] S. Fleury, M. Herrb, and R. Chatila. Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 842–848, 1997.

[15] B. Gerkey, R. Vaughan, A. Howard, and N. Koenig. The Player/Stage project. `http://playerstage.sourceforge.net/`, 2001.

[16] Groupe de Recherche en Robotique. Proteus: Platform for RObotic modeling and Transformations for End-Users and Scientific communities. `http://www.anr-proteus.fr/`.

[17] P. Hošek, T. Pop, T. Bureš, P. Hnětynka, and M. Malohlava. Comparison of component frameworks for real-time embedded systems. In *13th International SIGSOFT Symposium on Component-Based Software Engineering (CBSE 2010)*, pages 21–36, 2010.

[18] M. Klotzbücher, P. Soetens, and H. Bruyninckx. Orocos rtt-lua: an execution environment for building real-time robotic domain specific languages. In *International Workshop on Dynamic languages for RObotic and Sensors*, page 284289, 2010.

[19] Korean Institute for Advanced Intelligent Systems. OPRoS. `http://opros.or.kr/`.

[20] A. Mallet, C. Pasteur, M. Herrb, S. Lemaignan, and F. Ingrand. GenoM3: Building middleware-independent robotic components. In *IEEE Int. Conf. Robotics and Automation*, pages 4627–4632, 2010.

[21] R. Mirandola and F. Plášil. CoCoTA—Common Component Task. In R. M. Andreas Rausch, Ralf Reussner and P. František, editors, *The Common Component Modeling Example. Comparing Software Component Models*, volume 5153 of *Lecture Notes in Computer Science*, pages 4–15. Springer-Verlag, 2008.

[22] OASIS. Service Component Architecture. `http://www.oasis-opencsa.org/sca`.

[23] Object Management Group. Modeling and Analysis of Real-time and Embedded systems. `http://www.omgmarte.org`.

[24] Object Management Group. OMG. `http://www.omg.org`.

[25] Open Management Group. CORBA: DDS for Lightweight CCM. `http://www.omg.org/spec/dds4ccm/`.

[26] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *Proceedings of the Workshop on Open Source Software held at the International Conference on Robotics and Automation (ICRA).*, 2009.

[27] M. Radestock and S. Eisenbach. Coordination in evolving systems. In *Trends in Distributed Systems. CORBA and Beyond*, pages 162–176. Springer-Verlag, 1996.

[28] C. Schlegel and R. Wörz. The software framework SmartSoft for implementing sensorimotor systems. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 1610–1616, 1999.

[29] A. Shakhimardanov, N. Hochgeschwender, and G. K. Kraetzschmar. Component models in robotics software. In *Proceedings of the Workshop on Performance Metrics for Intelligent Systems, Baltimore, USA.*, 2010.

[30] R. Smits and H. Bruyninckx. Composition of complex robot applications via data flow integration. In *ICRA*, pages 5576–5580. IEEE, 2011.

[31] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison Wesley, 1998.

[32] H. Utz, S. Sablatnög, S. Enderle, and G. Kraetzschmar. Miro—Middleware for mobile robot applications. *IEEE Trans. Robotics and Automation*, 18(4):493–497, 2002.