Model-driven software development approaches in robotics research

Arun Kumar Ramaswamy, Bruno Monsuez, Adriana Tapus

Model-Driven Software Development Approaches in Robotics Research

Arunkumar Ramaswamy
[1]Department of Computer and System Engineering, ENSTA-ParisTech, 828 Blvd Marechaux, Palaiseau, France
[2]VeDeCom Institute, 77 rue des Chantiers, Versailles, France
arun-kumar.ramaswamy@ensta-paristech.fr

Bruno Monsuez
[1]Department of Computer and System Engineering, ENSTA-ParisTech, 828 Blvd Marechaux, Palaiseau, France
bruno.monsuez@ensta-paristech.fr

Adriana Tapus
[1]Department of Computer and System Engineering, ENSTA-ParisTech, 828 Blvd Marechaux, Palaiseau, France
adriana.tapus@ensta-paristech.fr

ABSTRACT

Recently, there is an encouraging trend in adopting model-driven engineering approaches for software development in robotics research. In this paper, currently available model-based techniques in robotics are analyzed with respect to the domain specific requirements. A conceptual overview of our software development approach called 'Self Adaptive Framework for Robotic Systems (SafeRobots)' is explained and we also try to position our approach within this model ecosystem.

1. INTRODUCTION

A robotic system is a software intensive system that is composed of distributed, heterogeneous software components interacting in a highly dynamic, uncertain environment. However, no systematic software development process is followed in robotics research. A major part of the research comprises of providing 'proof of concept' in order to substantiate the researcher's idea, for example, a robust path planning algorithm. Most of the time, they are developed from scratch or by using external code based libraries. Nevertheless, when such components are composed with other functional modules, the system does not exhibit the expected behavior. This has led to the increased time-to-market and large system integration efforts when such systems are to be used in safety critical applications.

In the last decade, the robotics community has seen a large number of middlewares and code libraries developed by different research laboratories and universities. They facilitate software development by providing framework infrastructure for communication (e.g. ROS [18]), real-time control (e.g. Orocos [3]), abstract access to sensors and actuators (e.g. Player [9]), algorithm reuse (e.g. OpenCV [2], PCL [23]), and simulation (e.g, Stage [9], Gazebo [14]). To a large extend, these frameworks have helped in rapid prototyping of individual functionalities, but system level analysis still remains an issue. System level properties such as response time, synchronization, deployment have been realized as accidental outcomes, rather than a design decision. It is high time that roboticists transform themselves as *system thinkers* in addition to being domain experts.

Motivated from the positive results from the application of Model-driven Software Development (MDSD) in other domains such as automotive, avionics, etc., the software engineering community in robotics is gradually moving in that direction [24]. Model-driven software development helps the domain experts shift their focus from implementation to the problem space. They are attracted by the fact that appropriately selecting the viewpoints and level of abstraction, the system can be analyzed more efficiently.

The model driven work flow cannot directly be applied in the robotics domain. The primary feature that distinguishes a robotic system with respect to other embedded system is that it operates in a highly dynamic environment. This unpredictability spans over various phases in software development - from requirement specification, system design, implementation, integration, and till it is deployed in real world scenarios. The system cannot be realized in an uni-directional process flow because the solution for a robotic problem cannot be finalized during the design time. It is because neither the problem space nor the target environment cannot be completely modeled as in embedded systems. Hence, the current trend is to generate a container with slots that allows the developer to insert the 'hand coded' functionalities. However, such an approach may result in inconsistencies with models since most of the code in robotics domain is through external class libraries, which do not follow any strict semantics. This is commonly termed as round trip problem in software engineering literature. Hence, a more iterative approach is required in the process.

This research paper has two objectives: the first objective is to identify the current MDE approaches in robotics and analyze how these approaches achieve general modeling related advantages and how effective are they in satisfying robot specific requirements. The second objective is to po-

sition our MDSD approach 'Self Adaptive Framework for Robotic Systems (SafeRobots)' in the model-driven ecosystem.

In our research paper [19], we have identified domain specific requirements for robotic software component. In general, the paper tries to answer the following questions:

1. How is domain knowledge modeled and how it is used in various phases of software development?
2. What are the common Separation of Concerns (SoC) that are relevant in robotics and how these SoCs are used for analyzing the desirable properties and facilitate system level reasoning?
3. How are the models used at run time?
4. How are the non-functional properties incorporated in the system?
5. How are the robotic component specific requirements, such as composability, compositionality, variability, technology neutrality, addressed in these systems?

The paper is organized as follows: Section 2 identifies the current MDSD approaches available in Robotics research and an overview of the methods used in each of these approaches is presented. In Section 3, some of the modeling features and robotic domain specific requirements are discussed with respect to the identified approaches. Finally, in Section 4 a conceptual overview of our MDSD approach is explained and Section 5 concludes the paper.

2. MDSD APPROACHES IN ROBOTICS

This section provides an overview of some of the MDSD approaches available in robotics. To the best of our knowledge, currently there are four model based approaches in robotics: BRICS Model [13], RobotML [7], SmartSoft [24], and V3CMM [1] approach. We have many component-based approaches in which modeling is mainly used for basic skeleton codes rather than using models as a developmental artifact.

2.1 BRICS Component Model

BRICS Component model [13] is built upon two complimentary paradigms - Model-driven approach and Separation of concerns. The syntax of the model is represented by Component-Port-Connector (CPC) meta-model and their semantics is mapped to the 5Cs - Communication, Computation, Configuration, Coordination, and Composition. The components represent computation and can be hierarchically composed to represent composite components. A composite component contains a coordinator whose is in charge of starting and stopping the computational components. The Ports represent the type of communication, for example: dataflow, events, service calls, etc. and the connectors connect two compatible ports. The components are configurated by using their visible properties, for example: maximum iterations for a planning algorithm. The compositional aspect concerns the interaction between the other 4 concerns. The BRICS approach is built using Eclipse framework and all the concepts are not integrated in the toolchain. The workflow can be roughly summarized as follows:

2.1.1 Workflow

1. Define the structural architecture by using components, port, and connectors.
2. Each complex component contains a coordinator that is defined using state machines.
3. Perform a M2T transformation to generate executable code (currently Orocos and ROS middlewares are supported).

2.2 RobotML

RobotML is a DSL for designing, simulating, and deploying robotic applications. It is developed in the framework of French research project PROTEUS [7]. The domain model consists of architecture, communication, behavior, and deployment metamodels. The architectural model defines the structural design using the CPC model. In addition, it also defines the environment, data types, robotic mission, and platform. The communication model associated with ports defines the type of communication - dataflow port or service port. The behavior model is defined using state machines. Specific activities are associated with states and transitions that are mapped to specific algorithms. The deployment model specifies a set constructs that define the assignment of each component to a target robotic middleware or simulator. The workflow is described below:

2.2.1 Workflow

1. Define the architecture using component-port-connector diagram.
2. Define the communication policy between components by setting the port attributes.
3. Define the behavioral model of each component using state machines.
4. Create a deployment plan by allocating the components to a middle or a simulator.
5. Execute M2T transformation to generate the executable code.

2.3 SmartSoft

SmartSoft [24] employs a model based approach in creating component skeleton (called *component hull* in SmartSoft terminology) that mediates the external visible services of a component and internal component implementation. The skeleton provides links to four different artifacts - internal usercode, communication to external components, platform independent concepts such as threads, synchronization, etc., and platform specific middleware, and operating system.

The communication between external services (interfaces to other components) and internal visible access methods (interface to user code inside component) is based on a set of communication patterns. A set of seven communication patterns are identified that are relevant to robotics systems: `send`, `query`, `push newest`, `push time`, `event patterns`, `state` and `wiring` patterns. The wiring pattern provides dynamic connection of components at run-time. These patterns provide required abstraction from implementation technologies with respect to the middleware systems. In order

to promote loose coupling between components, the objects are transmitted by value and the data are marshaled into a platform independent representation for transmission. The behavior of the component is specified by an automaton with generic states *Init, FatalError, Shutdown* and *Alive*. The Alive state can be extended by the user-defined states. The life cycle of the component is managed using these predefined states including the fault detection. The workflow can be summarized as follows:

2.3.1 Workflow

1. Create a Platform Independent Model (PIM) of component skeleton with stable interfaces to usercode, externally visible interfaces, and interfaces to SmartSoft framework. The component model contains explicit parameters and attributes, that finally need to be validated while generating platform specific models (for e.g., wcet: 100 ms [requirement]).
2. The platform specific information is then added to PIM and component attributes are refined (for e.g., wcet: 80 ms [estimation]).
3. In the deployment phase, a system is designed by wiring the components and system level parameters are extracted with the help of Platform Description Model (PDM) (for e.g., wcet: 85 ms [measurement]).
4. The timing parameters of system are exported to an external tool called Cheddar [25] to analyze for the timing analysis.
5. The PSI is generated from Platform Specific Model (PSM) using Model to Text (M2T) transformation.

2.4 V^{3}CMM

V^{3}CMM component meta-model consists of three complementary views: structural, coordination, and algorithmic views. The structural view describes the static structure of the components, coordination view describes the event driven behavior of the components and the algorithmic view describes the algorithm executed by each component based on its current state. The structural view consists of component, ports, interfaces and their interconnections. The coordination model is defined using UML state machines, while algorithmic view consist of UML activity diagrams. The workflow is described below:

2.4.1 Workflow

1. Define the common data types and interfaces.
2. Create the simple and complex component definitions.
3. Design the behavioral model of each component using the UML state machines.
4. Design the algorithmic models using the UML activity diagrams.
5. Link activities to state machines and state machines to components.
6. Execute M2M transformation to generate UML models and M2T transformation to generate the executable code.

Table 1: Feature Comparison of MDD Approaches

Feature	RobotML	SmartSt	BCM	V3CMM
Composability	×	×	×	×
Compositionality	×	×	×	×
Static Variability	✓	✓	✓	✓
Dynamic Variability	×	✓	×	×
Component Abstraction	✓	✓	✓	✓
Technology Neutrality	✓	✓	✓	✓
Knowledge Model	✓	×	×	×
System Reasoning	×	✓	×	×
Non-Functional Property Model	×	×	×	×

3. FEATURE ANALYSIS

An overview of features available in each approaches is depicted in Table 1. The detailed discussions are provided in the following sections:

Separation of Concerns

In MDD, the complexity of the software is managed using the mechanism called 'separation of concerns (SoC)'. Vertical SoC is built on multiple levels of abstraction. Model Driven Architecture (MDA), a model based architecture standardized by OMG specifies four abstraction layers - Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM), and Platform Specific Implementation (PSI). Horizontal SoC manages complexity by providing different overlapping viewpoints of the model at the same abstraction level. All of the MDD approaches in robotics primarily use only two vertical SoC - PIM and PSI. Horizontal SoC is seen only in one of the abstraction layer - PIM. A comparison of SoCs is depicted in Figure 1.

Composability

A model is said to be composable if its core properties do not change upon integration. In other words, the composability of a model guarantees preservation of its properties across integration with other components. A highly composable model can freely move around in the design space and assemble itself to a more complex structure that is semantically correct. However, semantic correctness of the composed models are not addressed in any of the approaches. It is very important in robotics since the components are highly heterogeneous in nature in terms of semantics and necessary for providing unambiguous interfaces. For example, the authors of [5], in their proposal of standardization, have provided approximately 24 different ways of representing geometric relation between rigid bodies and have proposed a DSL based on it [6].

Compositionality

Compositionality allows to deduce properties of a composite component models from its constituent components. It enables hierarchical composition of components and provides correctness of the structure. The reusability of a component is enhanced by providing compile-time guarantees by veri-

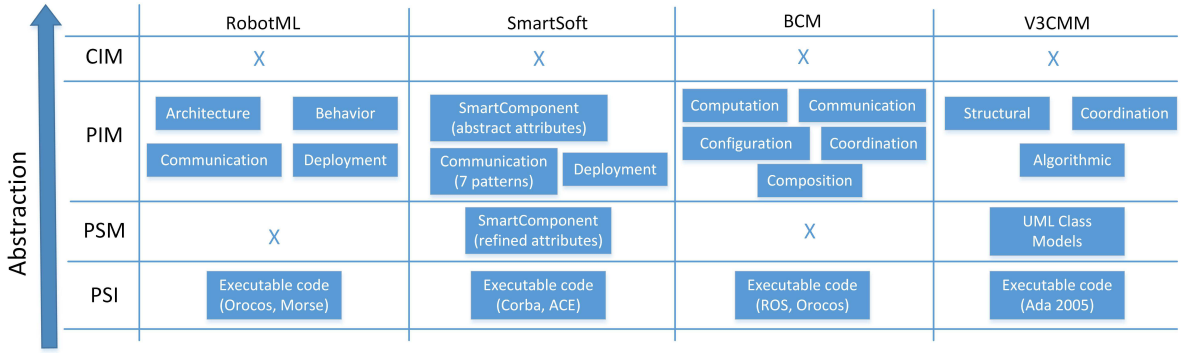

Figure 1: A comparison of vertical and horizontal separation of concerns

fying that formal parameters and actual parameters match regardless of the software module's location. The behavior of the reused components can be predicted in the new configuration and the result of the composition can be analyzed for anomalies. Only syntactic correctness of the architecture is addressed in all the approaches. It is worth mentioning that the works associated with Ptolemy [4] framework provide formal proofs for correctness when heterogeneous actors with different models of computation are composed. The major hindrance in robotics is the lack of standards, however, the robotics domain task force at OMG [15] and standard committee at IEEE Robotics and Automation society [21] is in process of standardization. The European project 'RoSta' provides standards and a reference architecture for service robots [22].

None of the architectures previously analyzed, currently provide support for the composability and compositionality properties. This is due to the fact that the component attributes are not explicitly modeled using formal methods.

Static and Dynamic Variability

Static variability is the configuration of the system and dynamic variability is related to the context dependent coordination of the components. Configuration defines which system can communicate with each other and coordination determines when such communication can occur. Configuration can be completely specified during design time while coordination is achieved by allowing variability during design time and run-time dynamic invocation. Static variability and limited context dependent dynamic variability is provided by the analyzed approaches. SmartSoft approach models variation points, context, Quality of Service, and Adaptation rules using Variability Modeling Language (VML) to support run-time variability for service robotics [11]. Brics approach uses feature models similar to the one used in Software Product Lin (SPL) to specify, implement, and constraint resolution of variabilities and provide graphical models for selecting possible configuration during design time [10].

Technology Neutrality

The component properties and specification should not depend on a specific technology. Software technology neutrality and hardware neutrality to some extend are achieved by many of the already available code-based frameworks, such as, ROS [18], Player project [9], etc. To a larger extend, all the MDSD approaches have achieved middleware independence by using various M2T approaches for generation executable code.

Modeling Domain knowledge

One of the primary focus of MDD is the separation of domain knowledge and implementation details. Among the compared MDD approaches, domain knowledge is explicitly modeled only in RobotML. In RobotML, the DSL is designed with the robot domain ontology as the backbone. In fact, the domain concepts that is required while designing the DSL is derived from the ontology. The ontology is used for two purposes - to normalize the robot domain concepts and to act as an inference mechanism during runtime [17]. However, it is not clear from the literature how the ontologies can effectively be used during model developmental and runtime models.

Round-tripping Problem

Round-tripping is a major concern of model-based system, especially if it has multiple abstraction layers and different horizontal separation of concerns. It is a major problem in analyzed approaches because separation of concerns is applied only at the model level. Restricting SoC only to models and in addition only to single abstraction levels worsens the round tripping effect and reduces the reusability. Providing SoC to models and code can support better traceability and system evolution. Aspect oriented modeling and template based techniques can be used to provide an integrated way of dealing with SoC. It will simplify the model development and transformation tasks [26] [16]. Approaches in RobotML and V3CMM provide only loose coupling among different viewpoints, for example in V3CMM approach, the uni-directional relationship between structural, coordinational, and algorithmic views have to manually be corrected if there is change in one of the viewpoints.

System level reasoning

SmartSoft and RobotML approach provide limited support to reason the timing analysis of the models at the system level. Properties such as WCET, Periods, etc, are provided as attributes to components. During the system deployment these properties are exported to an external tool 'Cheddar' for schedulability analysis. Semantic reasoning is not yet realized in any approaches because of the lack of standards.

Runtime Models

In robotics, the adaptation of the robotic system to the dynamic environments is embedded in the computational algorithms of the constituent systems. This severely limits the configuration space of such systems. Explicit modeling of the variabilities and variation points during the system design can help finding the best possible solution during runtime and can lead to the use of framework supported adaptation mechanisms. Hence, in order to achieve runtime adaptiation, explicit models of variation points, variabilities, context or environment and decision mechanisms should be supported by the framework. In [12] run time models are used for simple scenarios. The authors in [11] have demonstrated how SmartSoft framework with support of VML can be used for runtime adaptation for service robots.

Non-Functional Properties

Non-functional properties define how a functionality operates, for example, performance, availability, effectiveness, etc. QoS is the aptitude of a service for providing a quality level to the different demands of the clients. There is no general consensus in the community about the concepts of NFP and QoS. Non-Functional Requirements are not implemented in the same way as functional ones. NFPs are seen as by products when a functionality is implemented. The functional and non-functional attributes of the components are considered to make 'who does, what, and when' decisions depending on the operational context. However, non-functional properties are not given sufficient importance compared to that of the functional requirements during the developmental stages. However, none of the approaches analyzed provide explicit models for non-functional properties. SmartSoft uses NFPs as attributes to component model, but it is not formally modeled.

4. SAFEROBOTS FRAMEWORK

In this section, we propose and position our framework 'Self Adaptive Framework for Robotic Systems (SafeRobots)' in MDD ecosystem. Currently, it is in the conceptual and in initial stages of development. In our research paper [20], we have analyzed the common problems ecountered during software development in robotics research. These common issues can broadly be classified into three classes of problems:

Uncertain problem space: Ambiguity in requirements due to the desire to reuse the system across various applications. For example, in one of our previous projects, we decided to develop a vehicle tracking system that can be used for multiple application such as ground truth generation, autonomous driving, traffic detection. Later it was found that although the functionality is the same, the requirements for non-functional properties such as timing, confidence, and resolution were different for each scenarios. Since this was not formally captured in the system, the adoption of the tracking system in the target application was further delayed.

Large solution space: The availability of multiple algorithms for implementing a functionality is very common in the robotics domain, for example, for segmenting a point cloud that represents an outdoor environment, various methods can be used depending on the context, terrain type, etc. The decision on which algorithm to use is taken by the domain expert during the design phase without considering the operational profile of those functionalities and their prerequisites, run-time environment, potential interactions, etc.

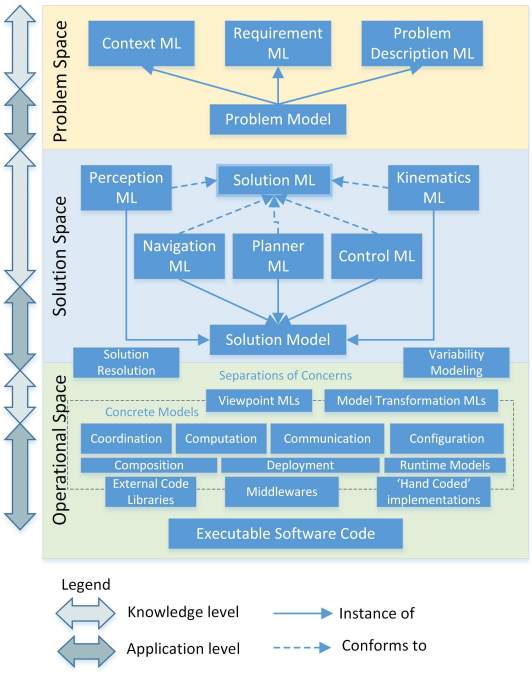

Figure 2: SafeRobots Framework: Ecosystem of Models and their relationships

Lack of design time context information: The developer of software component that realizes a functionality cannot anticipate all the use cases and his/her assumptions are not properly documented. Commonly, data flow driven models are used in robotics which cannot be effectively used of functional adaptation depending on the non-functional properties such as confidence, resolution of the data.

Incorrect level of Abstraction: External code libraries contain number of hard-coded magic numbers. They are either tightly bound to a particular sensor or to a specific scenario. Without properly documented meta-data, using such libraries hinders portability and reusablity.

In SafeRobots framework, the entire software development process can be conceptually divided into three spaces: problem space, solution space, and operational space. Each space can be further classified into knowledge level and application level. The complete ecosystem is illustrated in Figure 2. In problem space, the problem, requirements, and contexts are modeled using appropriate Modeling Languages (ML) using metamodeling approach. The solution model captures the large solution space that satisfies the problem model. The SSML proposed in our paper [20] provides abstract syntax with limited semantic content for modeling the solution. The syntactic and semantic enrichment is done by specific sub-domain modeling languages such as perception, navigation, control, kinematics, planner, etc. The DSLs proposed by the authors in [8] and [6] for kinematics and rigid body geometric relations can be easily integrated in the system. Operational space comprises of more concrete models that can be modeled as loosely coupled separation of concerns for in-depth analysis.

5. CONCLUSION

The paper identified currently available MDSD approaches available in the robotics research. A qualitative analysis of these approaches is performed with respect to the domain specific requirements and features. Most of the approaches are either in a development stage or in a conceptual state. Hence, quantitative analysis is currently not possible. However, it was found that many of the robotic specific requirements were not addressed. A conceptual overview of our MDSD approach, SafeRobots is explained by illustrating various metamodels and models involved in the proposed ecosystem.

6. REFERENCES

[1] D. Alonso, C. Vicente-Chicote, F. Ortiz, J. Pastor, and B. Alvarez. V3cmm: A 3-view component meta-model for model-driven robotic software development. *Journal of Software Engineering for Robotics*, 1(1):3–17, 2010.

[2] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'reilly.

[3] H. Bruyninckx. Open robot control software: the orocos project. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 3, pages 2523–2528. IEEE, 2001.

[4] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. 1994.

[5] T. De Laet, S. Bellens, R. Smits, E. Aertbeliën, H. Bruyninckx, and J. De Schutter. Geometric relations between rigid bodies: Semantics for standardization. *IEEE Robotics and Automation Magazine*, 2012.

[6] T. De Laet, W. Schaekers, J. de Greef, and H. Bruyninckx. Domain specific language for geometric relations between rigid bodies targeted to robotic applications. *arXiv preprint arXiv:1304.1346*, 2013.

[7] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane. Robotml, a domain-specific language to design, simulate and deploy robotic applications. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 149–160. Springer, 2012.

[8] M. Frigerio, J. Buchli, and D. G. Caldwell. A domain specific language for kinematic models and fast implementations of robot dynamics algorithms. *arXiv preprint arXiv:1301.7190*, 2013.

[9] B. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323, 2003.

[10] L. Gherardi and D. Brugali. An eclipse-based feature models toolchain. In *Proc. of the 6th Workshop of the Italian Eclipse Community (Eclipse-IT 2011)*, 2011.

[11] J. F. Inglés-Romero, A. Lotz, C. V. Chicote, and C. Schlegel. Dealing with run-time variability in service robotics: Towards a dsl for non-functional properties. *arXiv preprint arXiv:1303.4296*, 2013.

[12] J. F. Inglés Romero, C. Vicente Chicote, B. Morin, and O. Barais. Using models@ runtime for designing adaptive robotics software: an experience report. 2010.

[13] M. Klotzbuecher, N. Hochgeschwender, L. Gherardi, H. Bruyninckx, G. Kraetzschmar, D. Brugali, A. Shakhimardanov, J. Paulus, M. Reckhaus, H. Garcia, et al. The brics component model: A model-based development paradigm for complex robotics software systems. In *28th ACM Symposium on Applied Computing (SAC), Coimbra, Portugal*, 2013.

[14] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[15] T. Kotoku and M. Mizukawa. Robot middleware and its standardization in omg-report on omg technical meetings in st. louis and boston. In *SICE-ICASE, 2006. International Joint Conference*, pages 2028–2031. IEEE, 2006.

[16] V. Kulkarni and S. Reddy. Separation of concerns in model-driven development. *Software, IEEE*, 20(5):64–69, 2003.

[17] G. Lortal, S. Dhouib, and S. Gérard. Integrating ontological domain knowledge into a robotic dsl. In *Models in Software Engineering*, pages 401–414. Springer, 2011.

[18] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.

[19] A. Ramaswamy, B. Monsuez, and A. Tapus. Formal Models for Cognitive Systems. In *International Conference on Advanced Robotics (ICAR)*. IEEE, 2013.

[20] A. Ramaswamy, B. Monsuez, and A. Tapus. Solution Space Modeling for Robotic Systems. *Submitted to Journal of Software Engineering for Robotics (JOSER)*, March 2014.

[21] RAS-SCSA. Ieee ras standards committe. `http://www.ieee-ras.org/industry-government/standards`. Accessed January 30, 2014.

[22] RoSta. Robot standards and reference architectures. `http://www.robot-standards.eu/index.php?id=8`. Accessed January 30, 2014.

[23] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[24] C. Schlegel, T. Haßler, A. Lotz, and A. Steck. Robotic software systems: From code-driven to model-driven designs. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–8. IEEE, 2009.

[25] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. In *ACM SIGAda Ada Letters*, volume 24, pages 1–8. ACM, 2004.

[26] A. Solberg, D. Simmonds, R. Reddy, S. Ghosh, and R. France. Using aspect oriented techniques to support separation of concerns in model driven development. In *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, volume 1, pages 121–126. IEEE, 2005.