SafeRobots: A Model-Driven Approach for Designing Robotic Software Architectures

Arunkumar Ramaswamy*† and Bruno Monsuez* and Adriana Tapus*
*Department of Computer and System Engineering,
ENSTA-ParisTech, 828 Blvd Marechaux, Palaiseau, France
†VeDeCom Institute, 77 rue des Chantiers, 78000 Versailles, France
Email:{arun-kumar.ramaswamy; bruno.monsuez; adriana.tapus}@ensta-paristech.fr

***Abstract*—In this paper, a model-based framework called 'Self Adaptive Framework for Robotic Systems (SafeRobots)' for designing robotic software architectures is proposed. With the help of an example of designing an architecture for mobile robot mapping system, it will be shown how formal domain knowledge can be used at multiple abstraction levels in a model-driven engineering approach. The solution pool for a particular problem is modeled using Solution Space Modeling Language. An eclipse based tool-suite is developed for modeling the solution space and for deriving a concrete operational model by explicitly defining the non-functional properties of the constituent components.**

***Keywords*—Robot programming, Software engineering, Software architecture, Programming environments, Knowledge based systems**

I. Introduction

'Self Adaptive Framework for Robotic Systems (SafeRobots)' is a Model-Driven Software Development (MDSE) approach for designing software architecture for robotic system. In MDSE approach, abstract models are gradually converted into concrete executable models by a series of systematic transformational processes. Models are designed based on meta-models and domain-specific languages (DSLs).

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 1. In problem space, the problem, requirements, and contexts are modeled using appropriate Modeling Languages (ML) using meta-modeling approach. The solution model captures the large solution space that satisfies the problem model. A super language termed as Solution Space Modeling Language (SSML) 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 [1] and [2] 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.

The paper is organized as follows: Section II provides related works. Section III proposes a motivating example and discusses why knowledge models are necessary at various abstraction layers. Various processes involved in the workflow

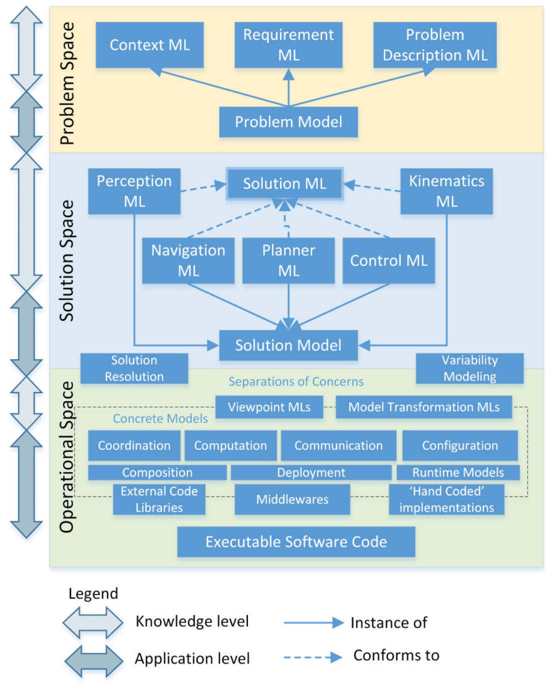

Figure 1: Model-driven Development Ecosystem Proposed for SafeRobot Framework

of SafeRobots framework is briefly described in section IV. Various frameworks involved in implementing the SSML editor are explained in section V-A. Furthermore, section V-B discusses the formal solution model for our example. And finally, section VI concludes the paper.

II. Related Works

Motivated by the positive results from the application of MDSE in other domains such as automotive, avionics, etc., the software engineering community in robotics is gradually moving in that direction [3]. MDSE helps the domain experts shift their focus from implementation to the problem space. By appropriately selecting the viewpoints and level of abstraction, the system can be analyzed more efficiently. Currently most of the MDSE based tools in robotics are in early stage of development or are in conceptual stage. The Smartsoft framework is based on a model-driven toolchain that support formal modeling of component skeleton that act as a wrapper around

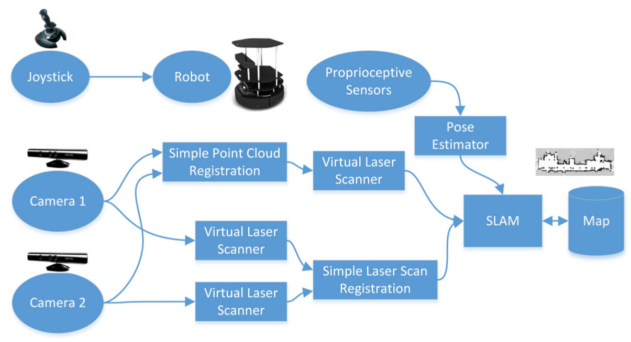

Figure 2: Informal Model for teleoperated mapping system

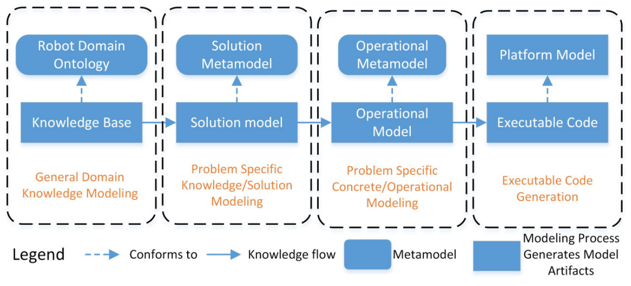
Robot Domain Ontology
Knowledge Base
General Domain Knowledge Modeling
Solution Metamodel
Solution model
Problem Specific Knowledge/Solution Modeling
Operational Metamodel
Operational Model
Problem Specific Concrete/Operational Modeling
Platform Model
Executable Code
Executable Code Generation
Legend
Conforms to
Knowledge flow
Metamodel
Modeling Process Generates Model Artifacts

Figure 3: General overview of modeling workflow

the user code [4]. The European project on Best Practices in Robotics (BRICS) provides guidelines and a framework to develop robotic components [5]. They are based on the separation of concerns between different aspects of Computation, Communication, Coordination, Configuration, and Composition. Currently it is in the developmental stage and only limited concepts have been integrated in the toolchain. RobotML, developed in the framework of the French research project 'PROTEUS' is a DSL for designing, simulating, and deploying robotic applications [6]. V^{3}CMM component metamodel consists of three complementary views: structural, coordination, and algorithmic views. But, it has not addressed any robotic domain-specific aspects [7]. However, none of the frameworks specifies any formal workflow and does not include semantic knowledge for system level reasoning.

III. Motivating Example

Consider the problem of designing a software architecture for a teleoperated mobile robot for creating a map of an indoor environment. The mobile robot is equipped with two 'Time of Flight' (ToF) cameras positioned at right angles to each other with overlapping 'Field of View' (FoV). The goal is to develop a SLAM based mapping system using the point cloud data from the two ToF sensors. Assume that a hypothetical library is also given that provides software code implementing the SLAM algorithm. The library also contains functions that can extract the points on a single plane that is at a specific distance from the robot base. When the raw point cloud is provided as input to the function, the resulting data is similar to that of the planar laser scanner. In addition, the library provides simple addition function that can combine two planar or multidimensional point clouds. In order to make the case simpler, we assume that the coordinate relation between the two camera frames are accurately known. This is for avoiding the complex point cloud registration methods.

Since all the computational algorithms are provided as black boxes as external code libraries, the primary task of the system designer is to create a structural architecture. Using pragmatic knowledge, this can be achieved by connecting each camera to the virtual laser scanner function and simply summing up the two planar point clouds and then fed to the SLAM algorithm along with the pose estimates. Another alternative is to combine the raw point cloud from the two cameras applying the provided registration function and then convert the resulting point cloud to the planar laser data. Either approach achieves the same functionality, but the timing properties will be different. Those non-functional properties cannot be predicted at this stage because we don't have any data regarding hardware platform in which it will be executed or whether all these computational components will be executed in a single platform or in a distributed fashion, and there is no information regarding the network bandwidth, latencies, etc. This will influence the rate at which the SLAM algorithm will be executed and it will affect the resolution and confidence level of the resulting map. An informal representation of these two solutions is illustrated in Figure 2. However, it is clear that we have used our pragmatism and past experiences to arrive at this mental solution model. This is in direct contradiction to the MDSE approach, since the knowledge used is in designer's mental form. This will not be possible for complex situations or when the system designer's knowledge is different from the computation developer's knowledge. Since such knowledge is not explicitly encoded anywhere in the design specification, this can hinder the system evolution for large scale projects.

IV. Modeling Workflow

The following section explains how the mental knowledge creation and reasoning process can be formalized in our SafeRobots framework. The entire workflow can be broadly classified into four processes as depicted in Figure 3. These are 1) General Domain Knowledge Modeling; 2) Problem-specific Knowledge or Solution Modeling; 3) Problem-specific Concrete or Operational Modeling; and 4) Executable Code Generation. Each sub-process is briefly explained in the following subsection.

A. *General Domain Knowledge Modeling*

At this process stage, the domain knowledge is modeled irrespective of the problem specification. The domain concepts are formally modeled using ontologies or DSLs. This level corresponds to Computation Independent Model (CIM) abstraction layer specified in the MDA architecture standardized by the OMG organization. The models at this level captures about the robotic domain specific concepts, meta-data about the computational algorithms, their structural dependencies, meta-data about the standard interfaces, etc, using conceptual spaces. The authors of [8] have proposed such a knowledge base for perception related knowledge base. They have demonstrated how to capture knowledge regarding processing steps for 'Robot Perception Architectures' (RPAs) using their DSL named as 'Robot Perception Specification

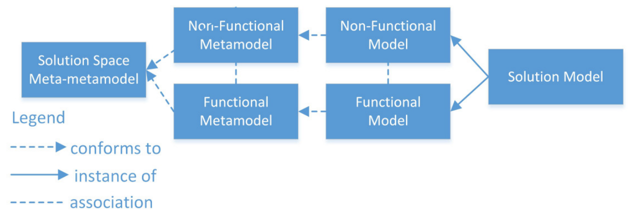

Figure 4: Meta-model and model relationships in SSML language

Language'. In the french research project PROTEUS [9], the robot domain knowledge is captured in the form of ontologies which later became the backbone for designing a modeling language called RobotML [6].

B. *Problem-specific Solution Modeling*

During the analysis of specific issues during software development during robotics research, one of the major class of problem is related to the large solution space, that exists in this domain, for example, for segmenting a point cloud that represents an outdoor environment, various methods can be used depending the context, terrain type, etc. Commonly, the decision on which algorithm to use is decided by the domain expert during the design phase without considering the operational profile of those functionalities and its prerequisites, run-time environment, potential interactions etc. In our approach, the solution space is formally modeled using 'Solution Space Modeling language' (SSML). To put in perspective, the solution space models can be viewed as a knowledge base, which is very specific to the problem in hand. It can be semi-automatically generated from the domain knowledge after applying the problem model as constraints. The formal representation of the solution space for the example discussed in Section III is explained in Section V.

C. *Problem-specific Concrete Modeling*

Problem Specific Concrete Models, also called as Operational Models, are a reduced subset of solution models. The reduction is carried out by considering the required system level non-functional properties such as timings, confidence, resolution levels, etc. There will be multiple solutions that satisfies the constraints and such situations are modeled as variation points that can be resolved after applying more concrete constraints or during runtime depending on the context.

D. *Executable Code Generation*

In this final process, an executable code is generated depending on the platform and the specified middlewares. Several Model to Text (M2T) techniques are available for applying this transformation, the details of which are beyond the scope of this paper.

V. Solution Model using SSML DSL

A. *SSML Editor*

The solution model conforms to the Solution Space Modeling Language (SSML) metamodel. It models the Functional

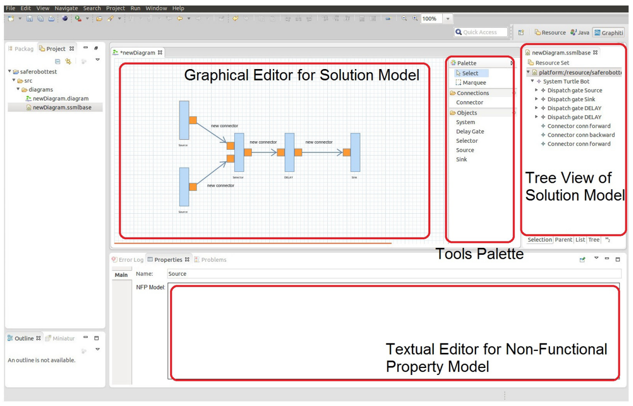
Graphical Editor for Solution Model
Tools Palette
Tree View of Solution Model
Textual Editor for Non-Functional Property Model
Palette
Select
Marquee
Connections
Connector
Objects
System
Delay Gate
Selector
Source
Sink
new connector
Name: Source
NFP Model
An outline is not available.

Figure 5: Editor for creating Problem Specific Knowledge/Solution Model

and Non-Functional Properties (NFPs) as illustrated in Figure 4. The syntax and semantics of SSML will be published in the extended version of this paper. The entire toolchain is based on Eclipse Modeling Framework (EMF) [10]. The metamodel is specified using Ecore meta-metamodel. The editor consist of a graphical section and a textual section as shown in the Figure 5. The functional architecture is modeled using the graphical editor and NFP model using textual editor. The graphical editor is built using the Graphiti [11], an Eclipse based graphical tooling framework. The essential building blocks such as Gates, Connectors, Source, Sink, etc, can be dragged and dropped from the tools palette. The connector is linked with the computational function from a code-based library. The abstract datatypes associated with the ports must be compatible with that of the interfaces provided by the computational functions. Apart from the functional link, the connector that represents a computation is also linked to the NFP model. The NFP model is modeled using a textual editor. The textual editor is built using Xtext framework [12]. Xtext is a framework for developing programming language and DSLs. It covers all aspects of a complete language infrastructure, from parsers, over linker, compiler, or interpreter and can be completely integrated to the Eclipse development environment.

B. *Solution Model*

The formal model of the solution space of the example discussed in Section III is shown in Figure 6. The model is realized using the modeling constructs providing by the SSML language. The connectors are linked to the respective computational functions denoted by the annotation associated with the connectors, for example, VLS.FP denotes the functional model and VLS.NFP denotes its NFP model of virtual laser scanner function. The NFP model is modeled using the textual editor. Sample textual models are shown in listing 1 and the $NFP\_Policy()$ functions are expressed in Object Constraints Language (OCL) [13]. During the resolution process, the model is converted into a graphical structure with gates as nodes and connectors as edges. The number of paths are then analyzed and the policies are applied for each element in the solution path and compared with that of the system policy. The solution path that passes all the policy constraints are then highlighted by the editor. The next step is the usage of the

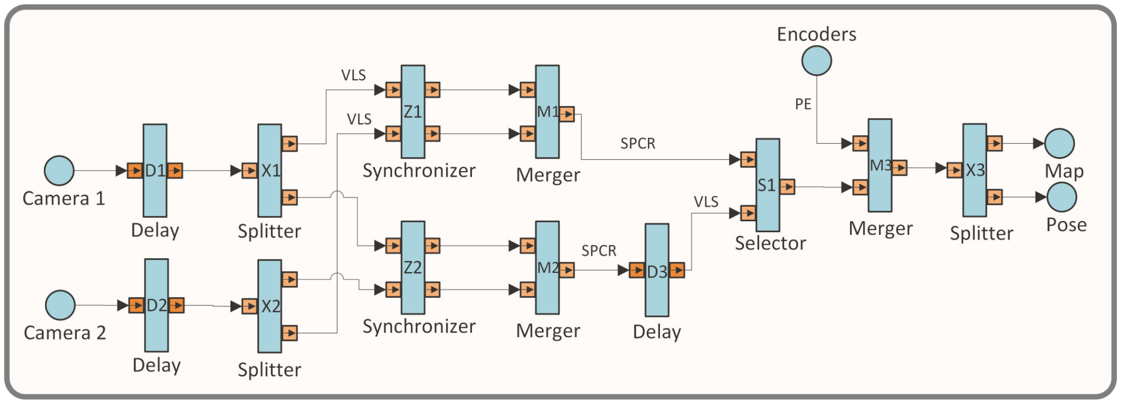

Figure 6: Formal solution model. The connectors are annotated with IDs that indicates the corresponding computations. The connectors without any annotation are simple direct connections. ID Notations: VLS-Virtual Laser Scanner, SPCR-Simple Point Cloud Registration, PE-Pose Estimator.

selected solution path for modeling in the operational space. The final model is then converted to the executable code based on the chosen platform or middleware. Currently, the tools have been tested only with ROS middleware and the generated artifacts are a set of code skeletons for computational nodes and a launch file for deployment.

```
NFP: VLS.NFP.exec_time
NFP_ATTRIBUTES: PCNP:PointCloud_No_Points, Dist:
     Distance_base;
NFP_POLICY: VLS.NFP.exec_time.policy();
---------------------------------------------
NFP: SPCR.NFP.exec_time
NFP_ATTRIBUTES: PCNP1:PointCloud_No_Points_1, PCNP2:
     PointCloud_No_Points_2;
NFP_POLICY: SPCR.NFP.exec_time.policy();
---------------------------------------------
IMPORT NFP VLS.NFP.exec_time, SPCR.NFP.exec_time, SLAM.
     exec_time, GATE_LIST.exec_time;
NFP: MappingSystem.NFP.exec_time
NFP_POLICY: MappingSystem.NFP.exec_time.policy() equals
     minimize(MappingSystem.exec_time);
```

Listing 1: Non-Functional Model of Virtual Laser Scanner (VLS), Simple Point Cloud Registration (SPCR) and that of the complete system

VI. Conclusion

The major contributions of this paper are formalizing four model creation processes for SafeRobots framework and an implementation of Eclipse-based tool for modeling the solution space. Even if the tool is in the prototyping stage, the initial results are very encouraging. More research is required on formal methods for specifying composable policies of Gates, NFP, and QoS in the proposed SSML language. Once the solution space of a problem is modeled, an appropriate solution or a set of solutions are selected depending on the application context and quality requirements. After this resolution, the next logical step is to model the solution more concretely in the operational space. A transformation model will facilitate this process of mapping to the operational space models, for example, discrete timing properties of gates can be employed for process allocation, selecting scheduling policies, etc. In a broad sense, SSML modeling primitives can be mapped to 5C approach proposed in BRICS project [5], for example, the mapping can be, connectors to computation, dispatch policies to communication, functional relationships to configuration, activation and deactivation of gates to coordination, and hierarchical composition of functional and non-functional aspects to composition and so on.

Acknowledgment

This research is funded by VeDeCoM Institute, a French automotive cluster on mobility research.

References

[1] 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.

[2] 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.

[3] 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*. IEEE, 2009, pp. 1–8.

[4] C. Schlegel and R. Worz, “The software framework smartsoft for implementing sensorimotor systems,” in *Intelligent Robots and Systems, 1999. IROS’99. Proceedings. 1999 IEEE/RSJ International Conference on*, vol. 3. IEEE, 1999, pp. 1610–1616.

[5] R. Bischoff, T. Guhl, E. Prassler, W. Nowak, G. Kraetzschmar, H. Bruyninckx, P. Soetens, M. Haegele, A. Pott, P. Breedveld *et al.*, “Brics-best practice in robotics,” in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. VDE, 2010, pp. 1–8.

[6] 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*. Springer, 2012, pp. 149–160.

[7] 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*, vol. 1, no. 1, pp. 3–17, 2010.

[8] N. Hochgeschwender, S. Schneider, H. Voos, and G. K. K. I., “Towards a robot perception specification language,” in *Proceedings of DSLRob-2013*, 2013.

[9] P. Martinet and B. Patin, “Proteus: A platform to organise transfer inside french robotic community,” in *3rd National Conference on Control Architectures of Robots (CAR)*, 2008.

[10] F. Budinsky, *Eclipse modeling framework: a developer’s guide*. Addison-Wesley Professional, 2004.

[11] M. Bauer, F. Filippelli, M. Gerhart, S. Kollosche, and M. Boger, "Concepts for the model-driven generation of visual editors in eclipse by using the graphiti framework."

[12] S. Efftinge and M. Völter, "oaw xtext: A framework for textual dsls," in *Workshop on Modeling Symposium at Eclipse Summit*, vol. 32, 2006.

[13] J. Warmer and A. Kleppe, *The object constraint language: getting your models ready for MDA*. Addison-Wesley Longman Publishing Co., Inc., 2003.