SafeRobots: A Model Driven Framework for Developing Robotic Systems

Arunkumar Ramaswamy[1,2] and Bruno Monsuez[1] and Adriana Tapus[1]

Abstract— **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. In this paper, we present the core concepts that drive our framework 'Self Adaptive Framework for Robotic Systems (SafeRobots)' for developing software for robotic systems. Two motivating examples are discussed: one discusses a system integration and knowledge representation problem, and the other explicates the issues associated with robotic system development in an industrial scenario. We also report on our progress on designing a meta-model based language - Solution Space Modeling Language, for problem specific knowledge representation.**

I. Introduction

In the past, the robotics community has been shown reluctance in adopting modern software engineering methods for developing software architectures. A major part of the robotics research concentrates on the delivery of 'proof of concepts' in order to substantiate the researcher's idea, for example, a robust path planning algorithm or a real-time collision detection system. Typically, these are developed from scratch or by using external code-based libraries. Nevertheless, when such components are merged and/or combined with other functional modules, the system does not always 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 research community has seen a large number of middlewares, code libraries, and component frameworks developed by different research laboratories and universities. They facilitate software development by providing infrastructure for communication (e.g., ROS [1]), real-time control (e.g., Orocos [2]), abstract access to sensors and actuators (e.g., Player [3]), algorithm reuse (e.g., OpenCV [4], PCL [5]), and simulation (e.g., Stage [3], Gazebo [6]). To a large extent, 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, flexibility, and 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.

This research is funded by VeDeCoM Institute, a French automotive cluster on mobility research. The research on vehicle tracking systems were conducted with the guidance from Javier Ibanez-Guzman at Renault, Research Division, France.

[1]Department of Computer Science and System Engineering, ENSTA-ParisTech, 828 Blvd Marechaux, Palaiseau, France

[2]VeDeCom Institute, 77 rue des Chantiers, 78000 Versailles, France

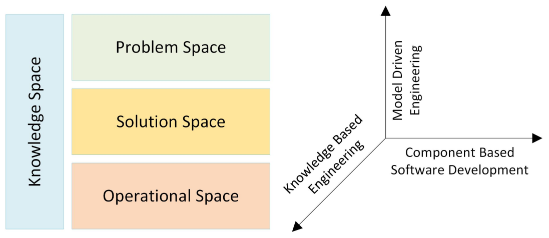


Fig. 1. Developmental Phases and Conceptual Paradigms in SafeRobots Framework

Motivated by the positive results from the application of Model-driven Software Development (MDSD) in other domains, such as automotive, avionics, etc., there is an encouraging trend in its adoption in the robotics domain [7]. 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 levels of abstraction, the system can be analyzed more efficiently. However, 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 systems 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.

In this paper, we address the following issues:

1) Identifying in a pragmatic way the domain specific problems and requirements of software development for robotic applications, that are not explicitly addressed in existing frameworks (Section II).
2) Identifying the core software engineering methods and developmental phases involved, on which the proposed framework is developed, in order to address the domain specific features (Section III & IV).
3) Developing a formal model for the problem specific solution space in order to capture multiple solutions for a given problem.

The rest of our paper is organized as follows: Two motivating examples are discussed in Section II: The first one focuses on the issues associated with developing a vehicle tracking system in an industrial context. The second

Preprint submitted to 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. Received February 6, 2014.

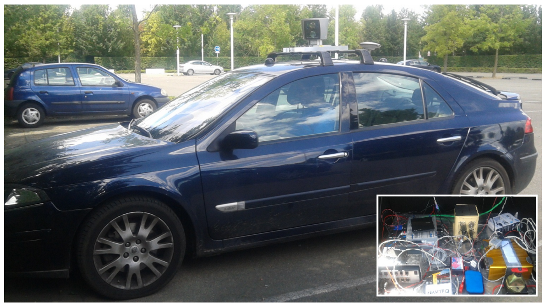


Fig. 2. Test Vehicle used for vehicle tracking experiment. Lidar and GPS are mounted on top of the vehicle and localization system and embedded computers are located in the trunk of the vehicle (see picture inset).

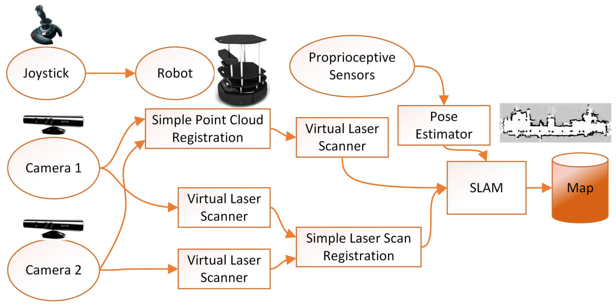


Fig. 3. Informal Model for a mapping system

example is on the design of a mobile robot based mapping system that poses some system integration challenges and exposes how we traditionally depend on the use of implicit mental knowledge to arrive to a solution. By addressing these identified issues, three orthogonal software engineering approaches and developmental phases on which our framework 'Self Adaptive Framework for Robotic Systems (SafeRobots)' is conceptualized, are explained in Section III and IV, respectively. In Section V, a meta-model language for representing the solution space in terms of functional and non-functional properties is presented and an example model based on this language is described in Section V-C.2. Related works are discussed in Section VII and finally, Section VIII concludes the paper.

II. Motivating Examples

A. Scenario No. 1

The first scenario is based on the issues encountered and lessons learnt during the system design, software development, and field experiments conducted during a research project at Renault, an automotive OEM. The objective of the project was to design and implement a vehicle tracking system using Velodyne HDL-64E lidar sensor. Velodyne lidar is a high definition laser scanning system that generates about a million of points per second using its rotating sensor head containing 64 semiconductor lasers. The lidar sensor is mounted on a passenger car as shown in Figure 2. The tracking system should detect vehicles in the environment and compute its state - 2D position and velocity. It was foreseen to use the system for the following scenarios: ground truth generation, real-time path planning system for autonomous driving, traffic surveillance, and for 3D mapping applications.

The technical problems encountered during the design and development of the tracking system can be broadly classified into the following three classes:

Uncertain problem space: The problem is not completely defined since there is an ambiguity in requirements due to the desire to reuse the system across various applications. Although the basic functionality is the same in all applications, 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: Multiple algorithms for implementing a functionality are available in the literature, for example, point cloud segmentation, multi-object tracking. 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, runtime environment, potential interactions, etc.

Lack of design time context information: The developer cannot anticipate all the use cases and his/her assumptions are not properly documented, for example, ground detection algorithm requires environment terrain conditions.

Level of abstraction: The system design was captured in a code-centric manner, that cannot provide the right level of abstraction so as to promote portability and reusablity.

B. Scenario No. 2

Consider the problem of designing a software architecture for a 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 an 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 a functions that can extract the points on a single plane (Virtual Laser Scanner) and for simple point cloud registration.

Since all the computational algorithms are provided as implemented software components, the primary task of the system developer is to design a data flow structure connecting these components. 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 it 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. Although, the two approaches seems to be very similar in functionality (creating a map), the system level emergent non-functional properties are different.

The non-functional properties of each software component, such as execution time, confidence level, cannot be predicted at this stage because we don't have any data regarding the 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 3. Nonetheless, it is clear that we have used our pragmatism and past experiences to arrive at this mental solution model. This integration method is highly non-deterministic, since the system integration for the same problem with the same 'ingredients' provides different results because the knowledge used is in the designer's mental form. The disparity will be more evident for complex situations or when the system designer's knowledge is different from the algorithm 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.

III. Conceptual Paradigms in SafeRobots Framework

Building on the identified problems and domain specific requirements, a conceptual framework called 'Self Adaptive Framework for Robotic Systems (SafeRobots)' is proposed here. This section provides an overview of the three orthogonal, yet complementary software engineering methods, on which the proposed framework is developed (see Figure 1 for an illustration).

A. Component Based Software Engineering

In Component Based Software Engineering (CBSE) approach, a software is developed using off-the-shelf components and custom-built components. A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only, and can be deployed independently and is subject to composition by third parties [8]. In our previous research paper [9], we have identified specific requirements of components, that are more relevant in robotics: composability, compositionality, static and dynamic variability, component abstraction, and technology neutrality. The main goal of CBSE is to manage complexity and foster software reuse by employing the *divide and rule* strategy. In order to promote reuse, the focus should be on the design and implementation of individual components and on the techniques that maximize component interoperability and flexibility [10].

B. Model Driven Engineering

In Model Driven Engineering (MDE), modeling techniques are used to tame the complexity of bridging the gap between the problem domain and the software implementation domain [11]. Although component based development and model driven engineering address complexity management, the former adopts a bottom-up approach, while the latter is more top-down in nature [12]. In MDE, the complexity of the software is managed by using the mechanism called 'separation of concerns (SoC)'. Conceptually, it can be classified into vertical and horizontal SoCs. Vertical SoCs are built on multiple levels of abstraction. Model Driven Architecture (MDA) [13], a trademarked name for MDE by Object Management Group (OMG), specifies four abstraction models - Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM), and Platform Specific Implementation (PSI). The computation independent model focuses on the environment of the system, the requirements of the system without any structural or processing details. The platform independent model focuses on the operation of the system while hiding the details regarding the platform, middleware, etc. The platform specific model is generated using PIM by adding platform specific details. The platform specific implementation is then generated by using appropriate Model to Text (M2T) transformations. Horizontal SoCs manage complexity by providing different overlapping viewpoints of the model at the same abstraction level. In [14], the authors suggest four horizontal SoCs for large scale distributed systems: coordination, configuration, computation, and communication. In summary, the main challenges in MDE are designing modeling languages at right abstraction level, modeling with multiple overlapping viewpoints, model manipulation, and maintaining viewpoints' consistency [11].

C. Knowledge Based Engineering

Knowledge Based Engineering (KBE) has the potential to change automated reasoning, methodologies and life cycle of software artifacts. In order to achieve that, domain concepts should be represented at various granularity levels in accordance with multiple abstraction layers. For example, in the case of self-driving cars, for system level reasoning, the knowledge required are regarding environment, socio-legal contraints, traffic rules, etc., and for platform independent layers, the knowledge required are about algorithmic parameters, structural configuration, semantics of data, etc. In general, information and knowledge can be represented in symbolic and non-symbolic representation. In the non-symbolic approach, a promising work is proposed by Gardenfors about representing conceptual spaces in the context of cognitive science [15]. Conceptual spaces are simple geometric representations based on quality dimensions, designed for modeling and managing concepts. The quality dimensions are used as framework to assign properties to objects and to specify relationships between them. The reasoning is mainly based on the distances between points in this conceptual space. In general, the smaller the distance between two objects, the more similar they are. For example, the distance from the concept 'canny edge detector' to the concept 'grayscale image' is smaller than to the 'rgb image'. However formalizing this approach with respect to software development is beyond the scope of this paper.

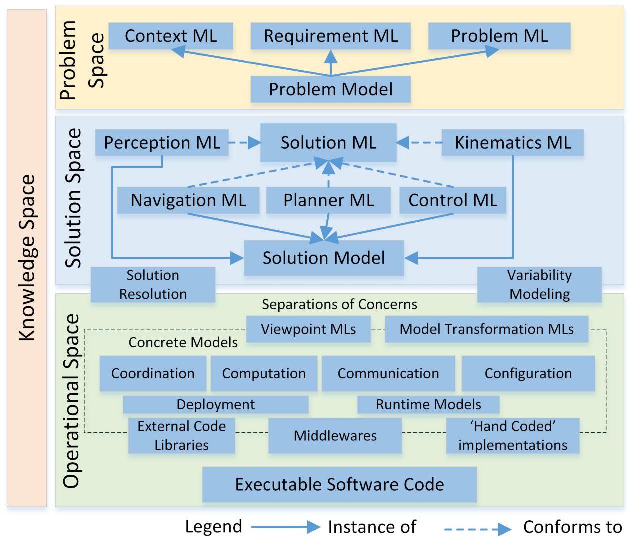


Fig. 4. Ecosystem of Models in SafeRobots Framework

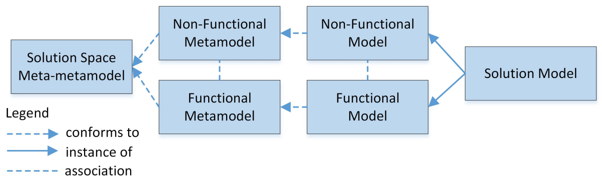


Fig. 5. Relationship between meta-models and models in Solution Space Modeling Language

IV. Developmental Phases in SafeRobots Framework

The different developmental phases, modeling languages, and models involved in the proposed framework are illustrated in Figure 4. The software development process in robotics can be conceptually divided into three spaces: problem space, solution space, and operational space, where each space is supported by the knowledge space. The following developmental phases are involved in the proposed framework:

A. General Domain Knowledge Modeling

The domain knowledge modeling in this phase is independent of the problem specification or application constraints. The domain concepts can be formally modeled using ontologies, Domain Specific Languages (DSLs), Knowledge graphs, etc. The models at this level captures the robotic domain specific concepts, meta-data about the computational algorithms and standard interfaces, their structural dependencies, etc. The domain knowledge complements the various application specific development process by providing a knowledge base for abstract concepts such as image, point clouds, links, joints, platform, etc.

B. Problem Modeling

In this phase of problem modeling, the application specific requirements, context or environment, are formally modeled. The functional and non-functional requirements are explicitly captured in the problem model.

C. Problem Specific Knowledge Modeling

Problem specific knowledge modeling or solution space modeling is designed with the help of functional requirements from the problem model as *constraints* applied to the domain model. In other words, a solution model captures multiple solutions for the given problem by considering only the functional requirements and given domain knowledge base. The strategy is postpone the decisions on non-functional requirements at a later stage, since such properties can be estimated only when platform specific decisions are made. In our approach, the solution space is formally modeled using 'Solution Space Modeling language (SSML)'. SSML is presented in detail in Section V.

D. Problem Specific Concrete Modeling

Problem specific concrete model, also called operational model, is a reduced subset of a solution model. The reduction is carried out by mapping non-functional requirements of a problem model, and system level and components' non-functional properties such as timings, confidence, resolution levels, etc. If there are multiple solutions that satisfy the constraints, they are modeled as variation points that can be resolved after applying more concrete constraints or during runtime depending on the context.

E. 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 Space Modeling Language

In this section, we present a meta-model based modeling language called 'Solution Space Modeling Language (SSML)' for modeling problem specific knowledge. In MDSE terminology, modeling languages are called meta-models and models as instances of meta-models. Meta-models are also models and the term 'meta' can be used recursively to represent the models at a higher abstraction level. By critical analysis of the issues and robot specific requirements, the desirable features for a solution space model are deduced: (a) It should be a graphical model that can be semi-automatically designed and visually inspected by humans, as well as machine readable; (b) It must be a hierarchical model that provides views at different granularity levels; (c) Non-Functional Properties (NFP) and Quality of Service (QoS) must be explicitly stated for functionalities; (d) It must capture the uncertainties in the problem space and must act as a reference model for developing concrete software models in the operational space. Equally important to the desirable features, the model should not include any implementation specific details, such as communication

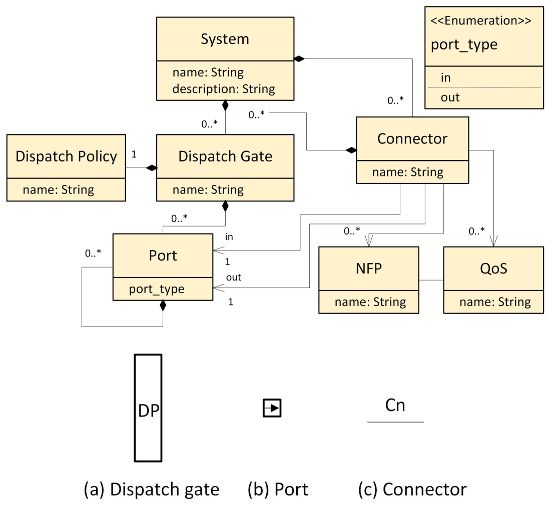


Fig. 6. Abstract syntax for modeling solution space. Dispatch Gate, Port, and Connector represents the functional aspect and NFP and QoS Profile represents the non-functional aspect. The Ecore diagram specifies the relationship between the modeling language constructs (top) and graphical representation of the primitive elements is also shown (down).

patterns, component packaging, and programming language specific constructs.

SSML is specified at two abstract levels as shown in Figure 5. Solution space meta-metamodel is at the highest level and functional and non-functional metamodels at the lower level in the hierarchy. Figure 6 shows the Ecore diagram and the graphical representation of the primitive elements. Ecore is a meta-language that is at the highest abstraction level in Eclipse Modeling Framework [16]. In natural language, the syntax can be explained as follows: the solution space consists of Dispatch Gates, Ports, Connectors as the primitive elements. A Dispatch gate consists of the number of ports and is associated with a Dispatch Policy. The ports can be 'in' or 'out' as indicated by the `port_type`. A connector is associated with two ports of which one should be an 'in' port and the other an 'out' port. A connector can have Non-Functional Property (NFP) and Quality of Service (QoS) associated with it.

The semantics of the proposed model is as follows: the dispatch gates represent basic operations for composing a different functional computational processes, such as selecting an appropriate data source for a computation, synchronization point, buffering, etc. The dispatch policy associated with it defines the operation of the gate. The data enters or leaves the gates through ports. Ports can be of type 'in' or 'out' depending on whether the data enters or leaves the gate. A connector connects two semantically compatible 'in' and 'out' ports. A Connector represents a functional computation and its quality aspects are represented by NFP and QoS. The SSML language provides only limited semantic constraints. The semantic enrichment of SSML is provided by specific sub-domain Modeling Languages (ML) such as Perception ML, Navigation ML, Control ML, etc., as illustrated in Figure 4.

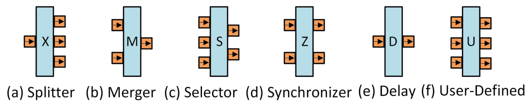


Fig. 7. Graphical representation of various dispatch gates

A. Functional Model

A functional model satisfies the behavioral requirements of the system. A behavioral or functional requirements are those requirements that specify the inputs (stimuli) to the system, the outputs (response) from the system, and the behavioral relationships between them [17]. The functional model in the solution space should not be interpreted as a data flow diagram, but as a relationship diagram between functional concepts. The model can be viewed as a knowledge diagram, that can visually be edited and understood by a human designer as well as by a computer. Hence the semantics related to the models which includes feedback connections, event based communications, etc. are handled at the operational model at the problem specific concrete modeling phase.

1) Functional Metamodel: The graphical contructs used in modeling the functional aspects is shown in Figure 7. The functional meta-model is implemented in Object Constraints Language (OCL) [18] and it imposes soft constraints on the number of ports associated with different dispatch gates. Soft constraints means that the dispatch policies are not concretely defined but only conceptual restrictions are only imposed. The intention is to facilitate automated reasoning by classfying the gates in the following categories.

a) Splitter gate consists of 1 input port and n output ports. It creates n splits/copies of the input data and transfers to its output ports.

b) Merger gate consists of n input ports and 1 output port. It merges the data from n input ports to the output port.

c) Selector gate consists of m input ports and n output ports. It selects m out of n input data.

d) Synchronizer gate consists of an equal number of input and output ports. It acts as a synchronization point between the different data streams.

e) Delay gate consists of one input and output port. It passes the data in the input port after a time delay and can also act as a data buffer.

f) User-defined gate does not have any constraints on the number of ports and dispatch policy.

B. Non-Functional Model

A non-functional model satisfies Non-Functional Requirements (NFR) and quality claims of the system. There is no general concord in the community about the concepts of NFP and QoS. In order to set a general consensus, based on empirical observations, the following assumptions are made: NFP are determined by a set of non-functional attributes (e.g., performance of a object classifier can be estimated by the time required for classification; and its efficiency by the rate of misclassification and number of object categories, etc.). QoS is a high level property for

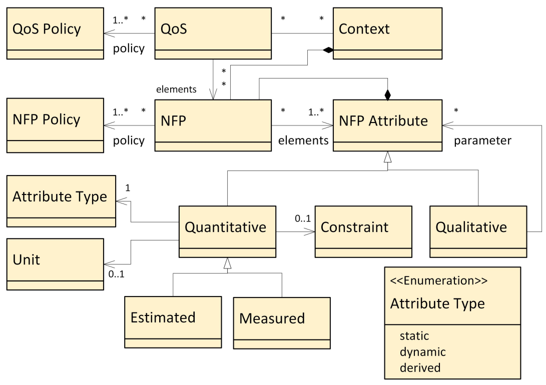
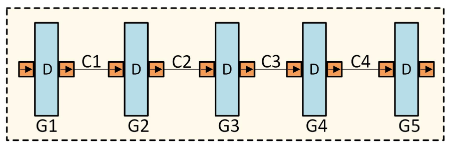
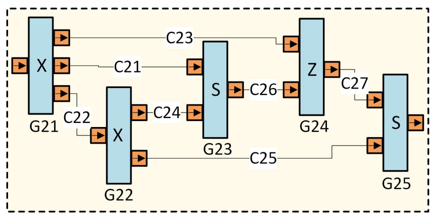


Fig. 8. UML Metamodel diagram for specifying Non-Functional properties.



C1:Data Preprocessing, C2: Segmentation, C3: Data Association, C4: State Estimation



C21: 2.5D Projection, C22: Ground Classifier, C24: 2D Projection, C25: Euclidean Clustering, C26:Image Segmentation, C27: Box Mapping

Fig. 9. Solution space model for vehicle tracking system. A high level model is shown (top), zero delay gates (G1-5) are inserted to separate the different computation functions. The connector C2 representing segmentation is expanded in the figure (bottom)

comparing a functionality in different contexts (e.g., QoS of a specific object classification algorithm is better in indoors as compared to outdoor environments). Policies associated with NFP and QoS determine how these properties are estimated from its constituent attributes. Policies are functions that act on a set of attributes, and define how these properties are estimated. Policies can be defined using logic systems, such as a first order logic, predicate logic, etc. [19]. In a nutshell, `NFP_Policy(NFP attributes)` defines NFP, `QoS_Policy(NFPs, Context)` defines QoS of a functionality.

1) Non-Functional Metamodel: Based on the assumptions made in Section V-B, an Ecore model for specifying the non-functional aspects is proposed in Figure 8. NFP and QoS are the first class entities in this metamodel. NFP has at least one NFP Policy and a set of NFP Attributes. Similarly, QoS has at least one QoS Policy and a set of NFPs as its attributes. NFP attributes can be Quantitative or Qualitative. Quantitative attributes can be directly measured or can be estimated. Quantitative attributes have metric units associated with it, for example: seconds for responsiveness, bits per second for throughput, etc. The quantitative attributes can be static, dynamic, or derived. Static attributes will not change during the course of system operation, for example, pixel density of a camera. Dynamic attributes can change during system operation, for example, frames per second of a camera. Qualitative attributes refer to discrete characteristics that may not be measured directly, but provide a high level abstraction that are meaningful in a domain, for example, reliability of a network channel. Sometimes a qualitative attribute needs some quantitative measures also, for instance, round robin scheduler with a refresh rate of 100ms. As explained previously in SSML meta-model description, such semantic enrichments are provided by the sub-domain modeling languages.

The following section describes how the proposed SSML is used representing the solution space for the scenarios discussed in Section II.

C. Solution Space Model for the example scenarios

1) Solution Model for Vehicle Tracking Example: This section models the solution space of a lidar based vehicle tracking problem using the proposed SSML. Figure 9 shows the high level model of a classical approach in tracking described in Section II-A. Four connectors, C1, C2, C3, C4, represent data preprocessing, segmentation, data association, and state estimation process and the zero delay gates are inserted to differentiate between these processes. The NFP model of data preprocessing computation represented by connector C1 is shown in Listing 1. This model consists of a single sequence of processes that does not have multiple solution paths at this hierarchical level. In this model, a solution is referred as an execution path or simply a path in the solution space model. In the textual form, it is represented as a sequence of labels indicating gates and nodes in that solution.

```
IMPORT CONTEXT vehicle;
NFP: C11.resolution;
NFP_ATTRIBUTES: ar:angular_resolution:est:static:deg,rps:
     rotations_per_second:msrd:static:Hz,tlppr:
     total_laser_points_revolution:msrd:static:int,pplpr:
     points_per_laser_per_revolution:msrd:static:int;
NFP_POLICY: resolution_lidar_policy();
```

Listing 1. Non-Functional Model of data preprocessing represented by connector C1 in Figure 9

The solution space for the segmentation process represented by connector C2 is expanded in Figure 9. The objective is to cluster a point cloud frame into smaller clusters in such a way that each cluster represents individual objects. There are multiple data processing steps and algorithms to achieve the goal. The segmentation solution model have captured multiple solutions. In general, a point cloud data can be segmented into clusters using two methods: (1) By directly processing the 3D data and clustering using some criteria, such as Euclidean distance or (2) By mapping the point cloud to a 2D image and performing segmentation in 2D space by using image processing techniques and then projecting back to the 3D space to compute the final point cloud clusters. In our research paper [9], we have detailed the properties of each solution for the vehicle tracking problem.

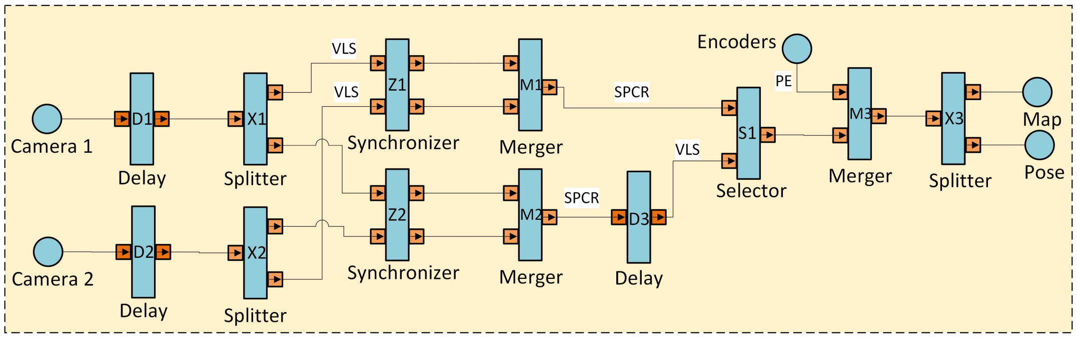


Fig. 10. Solution space model for mobile robot mapping system. The connectors are annotated with IDs that indicates the corresponding computations. The connectors without any annotation are simple direct connections. ID Notations: VLS-Virtual Laser Scanner, SPCR-Simple Point Cloud Registration, PE-Pose Estimator.

2) *Solution Model for Mobile Robot Mapping Example:*

The formal model based on SSML of the solution space of the example discussed in Section II-B is shown in Figure 10. The solution model captures the two solutions as previously discussed. 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.

VI. SafeRobots Implementation

The SafeRobots toolchain is based on Eclipse Modeling Framework (EMF) [16]. The metamodel is specified using Ecore meta-language. At this stage, the tool supports modeling the solution space based on the proposed SSML language. The SSML editor consists of a graphical section and a textual section as shown in the Figure 11. The functional architecture is modeled using the graphical editor and NFP model using textual editor. The graphical editor is built using the Graphiti [20], 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 [21]. 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. Currently, the tools have been tested only with ROS middleware and the generated artifacts are a set of code skeletons that links with computational classes and a launch file for deployment.

VII. Related works

Currently most of the MDSE based tools in robotics are in early stage of development or are in the conceptual stage. The

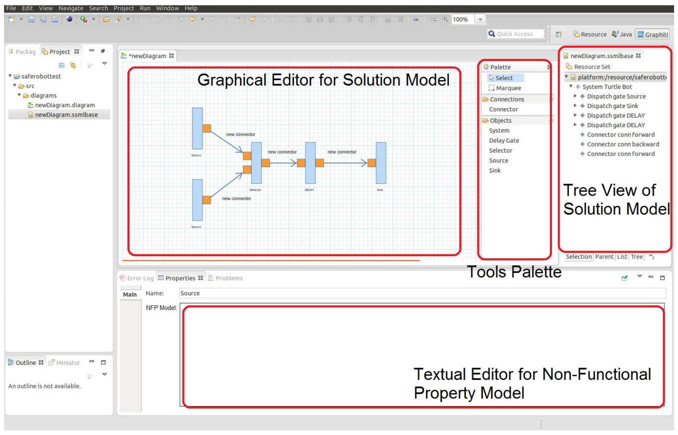

Graphical Editor for Solution Model
Tools Palette
Tree View of Solution Model
Textual Editor for Non-Functional Property Model
Palette
Select
Marquee
Connections
Connector
Objects
System
Delay Gate
Selector
Source
Sink
new connector
Main
Name:
Source
NFP Model
Outline
An outline is not available.

Fig. 11. Editor for creating Problem Specific Knowledge/Solution Model

Smartsoft framework is based on a model driven toolchain that support formal modeling of component skeleton that act as a wrapper around the user code [22]. The European project on Best Practices in Robotics (BRICS) provides guidelines and a framework to develop robotic components [23]. 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 [24]. V^{3}CMM component meta-model consists of three complementary views: structural, coordination, and algorithmic views. However, it has not addressed any robotic domain specific aspects [25].

Modeling solution space using SSML can be seen as a complementary approach to many existing methods in the robotic domain. Software product line approach is popular in software engineering community to enhance product quality and reduce developmental cost by promoting constructive reusability [26]. Recently, the author of [27] have adapted this approach in the robotic domain by providing feature resolution and transformation steps. Nevertheless, one should have already identified well defined boundaries for variation points and a reference architecture for adapting this approach. In addition, many DSLs are proposed in the robotics domain for deployment, simulation [24], component creation [28], etc. All these languages reside in the operational space. Our proposed SSML can complement and facilitate a smooth transition from problem space to operational space.

VIII. Conclusion

The research work presented in this paper focuses on systematic software development of robotic systems. Two scenarios - a simple robot integration example and a robotic functionality development in an industrial setup were discussed and several domain specific problems and requirements were then identified. The problems were classified into four categories: uncertain problem space, large solution space, lack of design time context information, and inability to decide on the right abstraction level. A conceptual framework, SafeRobots, was proposed based on three orthogonal software development approaches - Model Driven Engineering, Component Based Software Engineering, and Knowledge Based Engineering. The developmental phases in this framework can be classified into four phases - General Domain Knowledge Modeling, Problem Specific Solution Modeling, Problem Specific Concrete Modeling, and Executable Code Generation. A meta-model language for modeling problem specific knowledge, Solution Space Modeling Language, is proposed that explicitly separates functional and non-functional properties. Functional and Non-Functional Metamodel are proposed to model functional and non-functional aspects of the system. The relevancy of the model is demonstrated by applying it to model the solution space of vehicle tracking problem and the mobile robot mapping system.

However, the paper did not address 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.

References

[1] 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*, vol. 3, no. 3.2, 2009.

[2] H. Bruyninckx, "Open robot control software: the OROCOS project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3. IEEE, 2001, pp. 2523–2528.

[3] 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*, vol. 1, 2003, pp. 317–323.

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

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

[6] 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*, vol. 3. IEEE, 2004, pp. 2149–2154.

[7] 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.

[8] C. Szyperski, *Component software: beyond object-oriented programming*. Pearson Education, 2002.

[9] A. Ramaswamy, B. Monsuez, and A. Tapus, "Formal models for cognitive systems," in *International Conference on Advanced Robotics (ICAR)*. IEEE, 2013.

[10] D. Brugali and P. Scandurra, "Component-based robotic engineering (part i)[tutorial]," *Robotics & Automation Magazine, IEEE*, vol. 16, no. 4, pp. 84–96, 2009.

[11] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 37–54.

[12] M. Torngren, D. Chen, and I. Crnkovic, "Component-based vs. model-based development: a comparison in the context of vehicular embedded systems," in *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*. IEEE, 2005, pp. 432–440.

[13] M. Joaquin and M. Jishnu, *MDA Guide*, version 1.0.1 ed., Object Management Group, June 2003.

[14] M. Radestock and S. Eisenbach, "Coordination in evolving systems," in *Trends in Distributed Systems CORBA and Beyond*. Springer, 1996, pp. 162–176.

[15] P. Gärdenfors, *Conceptual Spaces: The Geometry of Throught*. MIT press, 2004.

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

[17] A. M. Davis, *Software requirements: objects, functions, and states*. Prentice-Hall, Inc., 1993.

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

[19] N. S. Rosa, P. R. Cunha, and G. R. Justo, "Process(nfl): a language for describing non-functional properties," in *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. IEEE, 2002, pp. 3676–3685.

[20] 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."

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] P. HEYMANS, J.-C. TRIGAUX, and F. E. Objectif, "Software product lines: State of the art," 2003.

[27] L. Gherardi, "Variability modeling and resolution in component-based robotics systems," Ph.D. dissertation, Katholieke Universiteit Leuven, Belgium, 2013.

[28] L. Manso, P. Bachiller, P. Bustos, P. Núñez, R. Cintas, and L. Calderita, "Robocomp: a tool-based robotics framework," in *Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, pp. 251–262.

Preprint submitted to 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. Received February 6, 2014.