An Extensible Model-based Framework Development Methodology for Robotic Systems

Arun Kumar Ramaswamy 1, Bruno Monsuez 2, Adriana Tapus 2

1 Renault Research Division, 1 Avenue du Golf, 78280 Guyancourt, France
2 Department of Computer and System Engineering, ENSTA-ParisTech, 828 Blvd Marechaux, Palaiseau, France

Abstract—Standardization, benchmarking, and formalization activities are being undertaken by many technical working groups and independent agencies such as, IEEE, ISO, and OMG, to promote Model-Based Software Development in robotics. Absence of integrated tools is the real barrier that exists between early adopters of such efforts and early majority of research and industrial community. The process of developing software frameworks and tools for designing robotic architectures is expensive both in terms of time and effort, and absence of systematic approach may result in ad-hoc designs that are not flexible and reusable. Therefore, within the context of architecture design, software development, and their supporting tools, a coherent practice is required for developing architecture frameworks. We believe that by making architecture meta-framework a point of conformance opens new possibilities for interoperability and knowledge sharing in the architecture and framework communities. In this paper, we tried to make a step in this direction by proposing a common conformance model and by providing a systematic approach that helps in specifying different aspects of software architecture development and their interplay in a framework.

Index Terms—Model-Driven Development (MDD), Architecture framework, Architecture Description Language (ADL)

1 INTRODUCTION

An architecture framework establishes a common practice for creating, interpreting, analyzing, and using architecture descriptions within a particular domain of application [1]. Although effective techniques for developing safety-critical software are well established, for example, in avionics industry, these techniques are in general designed for projects with long timescale and high staffing levels. It can be unsuitable for use without adaptation in the field of innovative robotics research, where timescale is shorter and, human resource and financial investment are typically much lower [2]. Frequently, in robotics domain, one has to deal with a wide variety of sensors and actuators with varying level of capabilities. Adding to this complexity of having heterogeneous hardware devices, robots has to deal with open-ended environment with limited resources [3]. The design, simulation, and programming of robotics systems is challenging as expertise from multiple domains needs to be integrated conceptually and technically. In addition, complex robotic systems typically involve software components that use a variety of mathematical models for problem solving, for e.g., state machines for robot control [4], stochastic Petri Nets for navigation [5]. As an emerging solution for handling complex and evolving software problems, several Domain Specific Languages (DSLs) have been proposed for specific functional domains in robotics [6]. They indeed help to raise the level of abstraction through the use of specific concepts that are closer to the respective domain concerns and facilitate validation and analysis [7]. A typical process for designing a robot architecture involves ‘mix and match’ of such architecture paradigms, mathematical models, DSLs, and implementation technologies. Without any common conformance model, this process is expensive both in terms of time and effort, and absence of systematic approach may result in ad-hoc designs that are not flexible and reusable. Systematic development process and detailed instructions for building such frameworks and supporting infrastructure have not been studied enough. Therefore, within the context of architecture design, software development and their support tools, a coherent practice is required for developing architectural frameworks.

The rest of the paper is organized as follows: A brief overview on robotic architectures is provided in Section 2. Section 3 introduces the notion of architecture framework and
related concepts. Section 4 details our approach of framework specification and different techniques involved. Section 5 details the proposed framework development approach using an example system in robotics. Section 6 discusses tools and suggests different approaches for the framework implementation. Related works is provided in Section 7, and Section 8 concludes the paper.

2 ARCHITECTURE MODELING IN ROBOTICS

The initial work on robot architectures began with two extreme approaches - one is based on deliberative sense-plan-act paradigm [8], and the other is a purely reactive subsumption architecture [9]. To take advantage of these two approaches, a number of hybrid architectures were proposed. A notable one is Gat’s three layer architecture that uses controller, sequencer, and deliberator layers to enable the robot to make high level plans and at the same time reactive to sudden events [10]. The Task Control Architecture (TCA), developed by Simmons, provides a general framework for controlling distributed robot systems. TCA is a high-level robot operating system with a set of commonly required mechanisms to support distributed communications, task decomposition, resource management, execution monitoring, and error recovery [11].

Cognitive architectures, e.g., ICARUS [12], use concepts from artificial intelligence and cognitive psychology to create and understand synthetic agent that support the same capabilities as human and thus makes the robots more pervasive in a social environment. All of these architectures can be broadly classified as conceptual architectures, though their level of abstraction with respect to the concepts in robotics domain is different.

The architectures that concern the execution and implementation aspects lie in the lower end of the abstraction axis. Modeling languages such as, UML, SysML, and Marte, model the system that is more closer to the software realizations. Their semantics are mostly implementation-specific and contain semantic aspects such as communication patterns and model of computation (MoC). There is another category of architecture that consists of conceptual architectures that are tightly bound to specific implementation models. Architectures, such as GenoM [13] and ACT-R [14], provide their software development kits (SDK) to design systems complying with those models. The main advantage of these models is that more stringent validation methods can be applied and can maintain traceability from domain concepts to its implementations. However, it takes significant effort to port from one implementation technology to another.

We can view this as a spectrum of models on an abstraction axis as shown in Figure 1. One can notice the concentration of architectures on both ends of the spectrum and the significant gap in the middle. One of the goals of our approach is to fill this gap by providing a common conformance model between conceptual architectures and platform specific implementation architectures.

3 ARCHITECTURE FRAMEWORK

An architecture framework is a collection of conventions, principles for the description of architectures established within a specific domain of application and/or community of stakeholders [1]. For example, NIST 4D/RCS is a robotic architecture framework developed by US Department of Defense for unmanned vehicle systems [15]. It provides a theoretical foundation for designing, engineering, and integrating intelligent systems software for unmanned ground vehicles. Necessary tools are also provided for building, testing, and evaluating 4D/RCS system to create the initial specification and finally, to generate generic templates for the RCS nodes. Generally, the architectures are specified using Architecture Description Languages (ADLs). ADLs provides notations and concrete syntax for characterizing software architectures. Typically, the framework also provides tools for parsing, viewing, compiling, analyzing, or simulating architectures specified in their associated description language. These tools can take the form of libraries of functions, specialized programming languages, or graphical editors, and make the constraints of the architectural framework explicit, while hiding the complexity of underlying concepts [16].

ISO and IEEE have produced a joint international standard, ISO/IEC/IEEE 42010 [1], System and software engineering -
Architecture description, to establish a coherent practice for developing architecture descriptions, architecture frameworks, and architecture description languages within the context of software life cycle and its processes. However, there are some arguments and discussions in the community regarding the definitions and their relationships between the concept of architecture description, framework, and meta-frameworks [17][18][19]. In order to position our work and motivate its importance, the relationships shown in Figure 2 are followed in our work. The relationships are inline with the recommendations provided by the international standard. Architecture conforms to an architecture framework and is specified using an ADL. Architecture addresses known concerns for known stakeholders for the system of interest. Architecture frameworks introduce a level of indirection such that the stakeholders for system architecture are not known when the framework is defined. However, common practice indicates that framework developers often have in mind, known or established stakeholders within the domain of the framework. These stakeholders motivate the set of architecture related concerns that the architecture framework will focus on. A conforming architecture identifies these concerns and it directly lead to a set of views to be included. Viewpoints govern these views and it establishes notations, model kinds, tools, techniques, and methods to be used while creating models. Model kind are conventions for a type of modeling, for e.g., data flow diagrams and state machines.

Model-driven software development (MDSD) promises to reduce errors and efforts needed for complex software projects by automated code generation from abstract software models. Based on our comparative survey on existing MDSD in robotics and qualitative analysis of their features, we found that many of the domain-specific requirements, such as architecture level analysis, non-functional property modeling, system configuration, deployment, knowledge representation etc., exits as isolated solutions [20]. It is hard to integrate the available solutions according to one’s requirement.

4 Architecture Framework Development Using SafeRobots Approach

In this section, we propose our approach for architecture framework development. Architecture frameworks can be seen as a collection of viewpoints and their relationships among them. The relation between the viewpoints is an important requirement for an analyzable architecture. The relationship can be loosely coupled (uni-directional) or tightly coupled (bi-directional). For example, a coordination view that models ‘when components should communicate’ should be aware of the configuration view that models ‘who communicates with whom’ of a system, while configuration view need not know about the coordination model. In order to facilitate the compliance with the requirements of ISO/IEC/IEEE 42010 standard, we specify the framework using templates given in Section 4.2. The concept of viewpoints and views, which are central to our framework development approach, is discussed in the following section.

4.1 Viewpoints and Views
A Viewpoint of a system is a work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns [1]. A viewpoint is a way of looking at systems; a view is the result of applying a viewpoint to a particular system of interest. In other words, a view is a description of the system relative to a set of concerns from a certain viewpoint. Similar to the use of modules and packages to manage the complexity of system, we employ viewpoints to manage the complexity of framework. Furthermore, the views can be seen as constructs for the management of architecture. A viewpoint isolates independently solvable aspect of a system in order to manage complexity. For example, a deployment view of the system addresses only the concerns related to initiating the execution of components in a particular order. The relationship between the aforementioned notions can be seen as follows:

viewpoints : framework :: views : architecture

Our approach do not enforce any particular views or viewpoints, but provides facilities for specifying viewpoints and further to create tooling support for view creation, interpretation, modification, etc. There are two different approaches for creating views: constructive and projective approaches. In the constructive approach, views of the system based on model kinds are created individually. These views are then synthesized to an overall model. Model correspondences are performed for integrating views from multiple model kinds. In the projective approach, the views are derived from the overall model. We strongly recommends the projective approach. This is to avoid the problem of view integration and view consistency. Since the semantics of the metamodel proposed in Section 4.3.3 is extensible, new model kinds can be extended and hence projective approach is a more convenient technique.

4.2 Framework Specification Templates
In order to claim the conformance of the architecture framework with the provisions of the international standard, templates with a set of information items or slots are provided. We use templates for specifying framework, viewpoints, and views, that the tool developer will use for implementing the tool. This document also serves as a reference in order to add new viewpoints and for implementing the relationship with the existing viewpoints. Framework and view template are given in Section 4.2.1 and 4.2.2, respectively. Viewpoint template is similar to view template with the addition of slots for describing concerns, stakeholders, and views, instead of model kinds and operations in view template. In order to broaden the scope, the international standard defines only
minimal requirements on architecture description to maximize the flexibility. The standard also states that many of these requirements can be met through a metamodel, a mapping of framework constructs to architecture models, a text narrative, or in some other manner.

4.2.1 Framework Template

**Framework name:** The name of the framework or phrase for identification.

**Viewpoint overview:** An abstract or brief overview of the framework and its key features.

**AMAL Formalism:** The specification of the framework in AMAL formalism, that is detailed in Section 4.3

**System stakeholders:** A listing of system stakeholders expected to be users of this framework

**Concerns:** A listing of the architecture-related concerns that are required for this framework.

**Viewpoints:** A listing of viewpoints available for the users.

**Examples:** This section provides examples for the framework developer.

**Notes:** Any additional information that users of this views might need or find helpful.

**Sources:** Identify the sources for this views, if any, including author, history, literature reference, etc.

4.2.2 View Template

**View name:** The name of the view or phrase for identification.

**View overview:** An abstract or brief overview of the view and its key features.

**Model Kinds:** Each model kind specified by this view is identified in this section. For each model kind used, describe the convention to structure the properties associated with the core elements of Architecture Modeling and Analysis Language (AMAL). The concepts of AMAL and its core elements are introduced in Section 4.3. Key modeling resources that the view makes available including how the model elements are visually rendered by the view and determine the vocabularies for constructing the view. These includes the tools for operations such model creation, deletion, making connections, editing properties, etc.

Sample scripts are provided to guide the tool developer. For example, the below script uses Acceleo Query Language (AQL) to find the source and target of a connector in a given view:

domain class: amal.connector
source finder expression:
[self.eGet('role')]->first().eInverse()]
target finder expression:
[self.eGet('role')]->last().eInverse()]

In our approach, we have used Sirius, that has inbuilt support for AQL scripts, for developing graphical workbenches [21]. Therefore, we have included AQL scripts in the templates to facilitate the tool developer while implementation. However, there is no restriction on the choice of model querying language.

A model kind may be documented in a number of ways such as:

1) by specifying a metamodel that defines its core elements and their relationships.
2) by providing a pseudo code that conforms to AMAL metamodel by specifying the associated properties.
3) via a language definition or by reference to existing modeling language.
4) or by combination of these methods.

**Operation on views:** Operations define the methods to be applied to views or to their models. Operations can be divided into categories [1]:

- **Creation methods:** These are the means by which the model elements represented by this view are created. This is usually in the form of process guidance and are specific to the tool used for creating viewpoint.
- **Interpretive methods:** These are the means by which view are to be understood by the system stakeholders.
- **Analysis methods:** These are used to check, reason and transform, predict, apply and evaluate results from this view.
- **Design methods:** These are used to realize or construct systems using information from this view.

**Examples:** This section provides examples for the framework developer.

**Notes:** Any additional information that users of this views might need or find helpful.

**Sources:** Identify the sources for this views, if any, including author, history, literature reference, etc.

In order to bring consistency in this specification and to promote reuse of infrastructure tools, we propose an Architecture Modeling and Analysis Language (AMAL) to formally specify the framework in the next section.

4.3 Architecture Modeling and Analysis Language

As illustrated in Figure 2, the model kind governs the view at the lowest level of abstraction. Example of models kinds that are common in robotics are state machines, Bayesian networks, control diagrams, and component-based software architectures.

*State machines* model the discrete behavior of a robot control system. It decides what activities must be running in the system in concurrent ways, and based on which events the system must switch its overall behavior to another set of concurrent activities. The structure of these switches is modeled by the states being connected through transitions [22].

A *Bayesian Network (BN)* is a directed acyclic graph $G = (V, E)$ with nodes representing a set of RVs $X1, X2, ..., X|V|$, and edges denoting conditional dependence relationships between random variables. BNs are widely used in robotics for localization, and for several learning algorithms [23].

*Control diagrams* such as Cartesian position control and other data flow models such as Simulink [24] consists of node
that represent certain functions and edges represent their input/output relationships.

Component based software architectures are approaches used to compose software systems from off-the-shelf and custom components [25]. In such models, nodes represent a piece of software with contractually specified interfaces that implement robotic functionality and the connector represents their interactions [26].

One can see that all the aforementioned models are some form of hierarchical component-connector diagrams with certain additional properties. The basic construct of AMAL formalism is based on this realization. The approach we have taken is to define a set of basic primitives upon which to construct different domain models such as, state machines and control diagrams. The core element of AMAL formalism and its semantics extensibility is discussed in the following sections.

4.3.1 AMAL Core Elements

The structure of AMAL is defined using four core elements: Component, Port, Connector, and Role. The semantics of the core elements are extended using properties as detailed in the next section. Figure 3 shows relationships between AMAL elements using Ecore diagram. The AMAL metamodel provides minimal syntactic rules using constraints, for example, a port cannot be connected to another port without a connector. The description of each core element is as follows:

1) Component: The component denotes the computation or a physical entity of the architecture. Components can hierarchically compose other components. A component can be atomic, representing an indivisible unit; or composite, representing a collection of components or an entire system itself.

2) Port: A Port represents the interface of the component. An interface is the interaction point of the component with the external environment. They are the external visible parts of the component that may facilitate data communication, monitoring, and reasoning of the component composition.

3) Connector: A connector represents the interaction between the components and are identified as the building blocks of the architecture. Depending on the domain semantics it can represent a data stream, event connections, state transitions, etc.

4) Role: A role represents the interface of the connectors similar to ports is for components. For example, a remote procedure call (RPC) connector consists of two roles: a callee and a caller role, an event broadcaster connector consists of one broadcaster role and arbitrary number of receiver roles. When connectors connect two components, roles are associated with compatible ports.

For example, Figure 4 depicts a system containing two nodes: talker and listener, communicating through publish-subscribe message passing paradigm. For illustrative purpose, each model element is labelled as \texttt{core_element: semantic_element}. In the domain that this system resides, a component is a node, a connector represents a publish subscribe communication protocol with associated roles taking the form of publish or subscribe accordingly. The ability to refine the semantics of the core elements is facilitated by \texttt{Property} element in AMAL.

4.3.2 Open Semantics Framework

Property associated with core elements of AMAL facilitates the annotation mechanism for extending the semantics of the model elements. For example, it can contain information on its
semantic information, non-functional properties, the graphical representation of the model element, etc. The contents of the property are not interpreted by AMAL. It has to be defined by the domain models. A deployment domain may use non-functional property associated with the component to dynamically allocate resources.

The bottom part of Figure 4 shows the textual representation of the publish-subscribe system. For simplicity, only the semantic information is shown as associated property in the model. The attributes of the Property element are shown using the syntax - Property name: type = value. It is to be noted that the textual form of the model is shown only for illustrative purpose. The model creation and manipulation is accomplished using the tools associated with specific viewpoints. However, it is important for the tool developer to understand how it will be captured in the model. For example, if the user click on a tool to create a node in the system, the tool will create a component with associated property with name as Node and value as semantic_element. A property can represent complex properties by composing more properties and can refer to other properties according to the AMAL metamodel. This mechanism along with viewpoints and tools provides the framework with different viewpoints, customized tools and rich visualizations.

4.3.3 AMAL Specification Formalism

We adopt a formalism to specify the framework in the template. The formalism defines the participating domain models, viewpoints, views, and their relationships.

Definition 1: An architectural framework $X$, that conforms to AMAL formalism is a tuple $\langle M_X, DM, IM, R_{AMAL}^{AMAL}, R_{AMAL}^{DM}, R_{AMAL}^{IM} \rangle$, where

- $M_X$ is the model defined in framework $X$.
- $DM$ is the domain model where the framework $X$ has conceptual relationships.
- $IM$ is the implementation model that framework $X$ supports.
- $R_{AMAL}^{AMAL}$ is a relation that associates AMAL model elements to the framework $X$.
- $R_{AMAL}^{DM}$ is a relation that associates model elements from $DM$ to AMAL model elements.

Definition 2: The viewpoints $V_n$, of an architectural framework $X$ is a set of tuple $\langle M_{V_1}, M_{V_2}, R_{V_1}^{V_2}, R_{V_2}^{V_1} \rangle$, where

- $M_{V_1}$ is the model defined in viewpoint $V_1$.
- $M_{V_2}$ is the model defined in viewpoint $V_2$.
- $R_{V_1}^{V_2}$ is a relation that associates model elements from $V_1$ to $V_2$.
- $R_{V_2}^{V_1}$ is a relation that associates model elements from $V_2$ to $V_1$.

The relationship between different domain models and implementation models in AMAL formalism is illustrated in Figure 5. Robotics domain is heterogeneous with domains ranging from conceptual domain such as perception, planning, control, decision making; computational domain consisting of discrete, continuous; software domain consisting of communication middlewares, operating systems, etc. Hence, framework should be extensible in order to incorporate different domain models. The composed domain models should be semantically compatible. For example, assume a model incorporates concepts from two domains $a$ and $b$. In domain $a$, the modeling element connector represents a computation process, and in domain $b$, the connector represents an instantaneous transition between two states. These two domains are semantically incompatible unless the conflict between them is resolved, say by assigning the computational process to component. AMAL formalism for an example application is given in the next section.

5 Case Study

In this section, we describe how an example framework, named ‘TrackX’, can be designed using our approach. TrackX is a framework for designing vehicle tracking applications using a LIDAR [27]. It uses concepts and algorithms from perception and object tracking domain. The framework shall support modeling a system using two complementary viewpoints: structural viewpoint and coordination viewpoint. The structural viewpoint models the interconnection of various computational algorithms and its communication aspects, and the coordination viewpoint models the coordination of these computations using state transition formalism. TrackX shall be able to generate ROS middleware based implementation. The topology of the planned framework is shown in Figure 6. The framework is specified using the templates discussed in
Section 4.2. We concentrate on the specification using AMAL formalism and the templates are not given to conserve space.

**AMAL Formalism:** The TrackX framework in AMAL formalism is defined as:

\[ \text{TrackX}_{AMAL} = (M_{TrackX}, PM, CM, RM, R_{AMAL}^{TrackX}, R_{AMAL}^{CM}, R_{AMAL}^{PM}, R_{AMAL}^{SV}, R_{AMAL}^{CV}). \]

With reference to the required viewpoints, the relationships are defined as:

\[ \text{TrackX}_{AMAL} \Rightarrow (M_{SV}, M_{CV}, R_{SV}^{AMAL}, R_{CV}^{AMAL}, R_{SV}). \]

A pictorial representation of the relationship between different domains is shown in Figure 7. Each element is explained as follows:

- **M_{TrackX}** denotes the model of the system that conforms to the AMAL metamodel.
- **M_{SV}** and **M_{CV}** denotes the model elements pertaining to the Structural View (SV) and Coordination View (CV) of the system, respectively.

**Perception Model (PM):** Perception model captures the conceptual knowledge of the perception domain. It contains abstract knowledge regarding various computational algorithms, its dependencies, etc. It is modeled using Solution Space Modeling Language (SSML), which is result of our previous research work [28]. The domain model specified in SSML contains meta-data on algorithms, their execution sequence, non-functional properties, expected quality of service, their constraints, etc.

**Communication Model (CM):** Communication models capture the knowledge regarding several patterns that cover the communication requirements for component interactions. In this example, we use the communication patterns proposed by the authors of [3]. It consists of a minimal set of communication patterns required by robotic software component interaction: send, query, push_newest, push_timed, and event.

**ROS Model (RM):** represent the implementation model of ROS middleware. The ROS specific concepts is specified in the form of Ecore model.

\[ R_{AMAL}^{SV} \] denotes the relationship between AMAL metamodel and domain model. For example, \( R_{AMAL}^{CM} \) represents the relation between communication model and the elements in the AMAL model. For example, \( R_{AMAL}^{CM} \) maps the communication patterns in CM domain to the core element: **Connector** in the structural view. It also details the the number of roles that each connector is associated with, depending on the pattern. For example, a connector that represent push_timed can shall have 1 publisher role and ‘n’ number of subscriber roles. Some approaches to implement such relationships are discussed in the next section.

The tool developer implements the framework according to the specification. Figure 8 shows the screenshot of the developed tool. A high-level structural model and its coordination model of a vehicle tracking system is shown in the screen shot. The component denotes a computational process and can be hierarchically composed, i.e., perception component can be expanded to represent the system a lower abstraction level with more detailed view of algorithms involved. The connector in the SV represents the communication pattern in CM between the components. The coordination view models the system behavior using states and transitions. The states represents the activation/deactivation of computational processes in structural view and transition coordinate the state changes. The tools palette shown in the right side of the workbench execute the AQL scripts, from view template specification (see Section 4.2.2), for creating or deleting the model elements. During the code generation, ports and connectors in SV generates ROS messages and topics, respectively. The next section provides guidelines and some implementation suggestions for the tool developer.

6 FRAMEWORK IMPLEMENTATION PROCESS

In this section, we detail different stages in which the framework developer can implement and deploy the framework workbench. Eclipse Modeling Framework (EMF) is used for implementing the framework. Eclipse is an open platform and it is designed to be easily and infinitely extensible by third parties. From a model specification point of view as described in XML Metadata Interchange (XMI), EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Associated with EMF, there is an ecosystem of plugins that assist the developer during different phases of model based tool development [29]. It is to be noted that our framework specification approach is independent of the implementation environment.

The first step is to specify the framework based on AMAL formalism and document using the given templates. This
process identifies the domains involved, required viewpoints, target platform and middleware, etc. The next step is to create models of the identified domains involved in the framework. This can be performed in two ways:

1) The domain models are modeled using Ecore based metamodel or using our SSML specification. The tool developer maps these model elements with the respective elements in AMAL metamodel. For this procedure, the developer creates model transformation templates to transform the model to our AMAL compatible model. However, we recommend the second procedure, as it eliminates the need to create a model transformation tool.

2) By using the model creation tools in graphical workbench to create model elements directly based on AMAL metamodel. For example, when the user create a component that represents a state machine, the property of the newly created component will be having name as `semantic_element` and value as `state_machine`.

Based on the artifacts generated in this stage, the developer creates appropriate Eclipse plugins to implement the framework. Implementing the graphical modeling workbench is the process that consumes most of the time according to our experience. We recommend Sirius for developing graphical workbenches [21]. Sirius have inbuilt support for implementing the viewpoints, different graphical options, and validation mechanisms.

It must be noted that all of the functionalities of Eclipse are located in different plugins. In order to promote reusability, the developer should to make it more modular in the form of several plugins. A plugin is a small unit of Eclipse platform that can be developed separately. For example, if the framework has a structural viewpoint and behavioral viewpoint, these will be implemented as two separate plugins. The specific viewpoint will be available to the user only if the corresponding plugin is installed. This also helps in mix and match of frameworks according to the user’s need. The plugins can also be provided in the form of a repository, from where the user selects and installs according to the requirement. The main constraint is that necessary relationship and model transformations should be accordingly available in the tool. At this stage, the complexity lies in designing a code generator that can handle semantic composition. This aspect has to be further explored. The authors of [30] have proposed an approach for synthesising code from integrated models.

7 RELATED WORKS

Our approach of providing common syntax with semantic content, whose semantic enrichment is provided specialized sub-domain specific language is conceptually related to that of ACME [31]. ACME is an interchange language for software architecture that provides structural core that represents commonalities between various Architectural Description Languages (ADL) [32]. ACME uses annotations to add semantic information by sub-languages. However, their objective is to serve as a common representation for software architectures and that permits the integration of diverse collection of independently developed architecture analysis tools. The Architecture Analysis and Design Language (AADL) is a modeling language standardized by Society of Automotive Engineers (SAE) to specify and analyze software architectures for complex real-time embedded systems.

The authors of [33] introduced Meta Architecture Description Language (MADL) to define, comment, document, compare architectures, in particular semiformal architectures. Their main proposal is for unifying ADLs in the context of software architectures and to provide reflexivity in architecture metamodeling. However our work is at the framework level.
and we introduced common primitive elements with semantic extensibility to manage complexity of multiple viewpoints and views. Rich Hilliard introduced decorative stance as an alternative to constructive views in frameworks. Our work is more similar to this approach. The approach is to decorate a primary representation with attributes pertaining to other concerns, rather than separate concerns. This is to address multiple-view problem in architectures, that lead different specification describe different, but overlapping issues. In another work, the authors have shown how behavioral semantics can be integrated in a structural formalism [34]. Our method does not enforce any such specific views and do not constraints on any such model kinds. Recently, a similar approach for DSLs in robotics is proposed by the authors of [22]. They have used hierarchical hypergraphs to model the structural properties of robotic systems and to enrich such graphs with domain specific attributes.

8 Conclusion
In this paper, we have formalized the robotic framework design and development process in order to build custom frameworks and integrate different architecture patterns in architecture development. The specification of framework, viewpoints, and views using the templates will meet the requirements of the ISO/IEC/IEEE 42010 standard. Making architecture meta-framework a point of conformance opens new possibilities for interoperability and knowledge sharing in the architecture and framework communities. We tried to make a first step in this direction by proposing common model and provided a systematic approach that helps in specifying different aspects and their interplay in a framework. The multi-domain architecture modeling helps to build integrated intelligent robotic systems. The uniform definition of architecture viewpoints and coordinated collections of such viewpoints in an architecture workbench can promote reuse of tools and techniques. In this direction, we have presented a method to model architecture framework based on heterogeneous architectural paradigms. The semantic extensibility feature helps to refine the semantic knowledge, and to specify inter-domain relationships. Another potential benefit is that the look and feel of the graphical interfaces will be the same irrespective of the framework. The tools provided with the frameworks provides homogeneous user interfaces and thus promoting faster adoption among users.

References


Arunkumar Ramaswamy is a researcher at Renault Research Division in France. He received his Ph.D. in Computer Science from ENSTA ParisTech (2017). His research interests include autonomous driving, mobile robots, robot architectures, computer vision, human-robot interaction and software engineering. Currently, he is investigating how model-driven approaches can be applied in software development for autonomous vehicles. Arunkumar received his dual Master degree in Robotics from Ecole Centrale de Nantes in France (2012) and Warsaw University of Technology in Poland (2011). He received his B.Tech degree in Electronics and Communication Engineering from Amrita University in India (2008).

Bruno Monsuez graduated in 1989 from Ecole Polytechnique, received a Ph.D. in Computer Science from the Ecole Polytechnique in 1994. He is now Director of the Computer and System Engineering Department at ENSTA ParisTech. His current research interests are focused on developing and enhancing hierarchical compositional models that can be used to model complex cyber-physical systems, mostly for safety critical application as well as on designing system architectures based on design patterns that can be proved correct by composition. He also conduct researches on novel formal verification techniques that allow a co-jointly verification the functional and non-functional properties of all the components of a system. He served on PCs and as PC chair for numerous international workshops and conferences. He is the steering committee chair of Int. Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS).

Adriana Tapus is a Full Professor at ENSTA-ParisTech since May 2009. She received her Ph.D. in Computer Science from Swiss Federal Institute of Technology, Lausanne (EPFL) in 2005 and her degree of Engineer in Computer Science and Engineering from Politehnica University of Bucharest, Romania in 2001. She worked as an Associate Researcher at the University of Southern California (USC), where she pioneered the development of socially assistive robotics, also participating to activity in machine learning, human sensing, and human-robot interaction. Her main interest is on long-term learning (i.e. in particular in interaction with humans), on-line robot behavior adaptation to external environmental factors, and modeling non-functional properties for human-machine systems. She received the Romanian Academy Award for her contributions in assistive robotics in 2010.