A UML 2-compatible language and tool for formal modeling real-time system architectures

Pietro Colombo
Dipartimento di Informatica e Comunicazione
Università degli Studi dell'Insubria
Via Mazzini 5
21100 Varese, Italy
pietro.colombo@uninsubria.it

Matteo Pradella
CNR Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni
Via Ponzio 34/5
20133 Milano, Italy
pradella@elet.polimi.it

Matteo Rossi
Dipartimento di Elettronica ed Informazione
Politecnico di Milano
Via Ponzio 34/5
20133 Milano, Italy
rossi@elet.polimi.it

ABSTRACT

ArchiTRIO is a formal language, which complements UML 2.0 concepts with a formal, logic-based notation that allows users to state system-wide properties, both static and dynamic, including real-time constraints. This paper summarizes the ArchiTRIO approach, and presents the core elements of a tool supporting it, called ArchiTRIDENT, which is currently under development. This tool is a plugin of the TRIO-based editing and verification TRIDENT tool suite.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*Tools*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Object-oriented design methods*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods*; D.2.11 [**Software Engineering**]: Software Architectures—*Languages*

Keywords

Formal methods, system architectures, real-time systems

1. INTRODUCTION

While the Unified Modeling Language (UML, [10]) has become the *de facto* standard for software and system modeling in industrial practice, its applicability to critical systems, where precise and rigorous designs are of the utmost importance, has been hampered by its lack of formality. This has not changed with the release of the 2.0 version of the UML specification [10]: UML 2.0 includes previously missing constructs (e.g., component, connector, port) that are necessary for describing system architectures; however, it does not yet address issues that are crucial in the development of critical (e.g. real-time and embedded) systems, as it lacks a formal semantics, and does not include a precise notion of time.

SAC'06 April 23-27, 2006, Dijon, France

ArchiTRIO [13, 12] is a language designed to instill formality in a subset of the UML 2.0 notation [10]. The key idea at the core of the development of ArchiTRIO is to *complement* the UML notation of class and composite structure diagrams [10] with a temporal logic-based notation. The ArchiTRIO approach to system modeling falls essentially in the category of lightweight formal methods [14]; more precisely, ArchiTRIO allows developers to use standard UML 2.0 notation to describe non-critical aspects of systems, but it also offers a complementary formal notation, fully integrated with the UML one, to represent the system aspects that require precise modeling. In this approach, a user who at first does not need full-blown ArchiTRIO can start by drawing bare UML class diagrams, and only later, when the need arises for clarity and precision (in particular for temporal constraints), introduce ArchiTRIO-specific notation.

ArchiTRIO is based upon few selected UML 2.0 constructs especially suited for describing architectures; it gives them a formal meaning, and precisely defines their composition. [13] presents the principles behind the language and suggests guidelines for its application, while [12] deals with its formal semantics. Both articles compare the ArchiTRIO approach with existing ones, especially with the Object Constraint Language (OCL, [9]), and highlight the differences and the novelties of the former.

A tool for the creation and management of ArchiTRIO models is currently being developed as part the *TRio Integrated Development EnvironmenT* (TRIDENT), an open environment based on the Eclipse platform [5]. The tool is built around a core that includes a meta-model of ArchiTRIO specifications; it is designed to be interfaced with external UML 2.0-compliant tools, to allow for the importing of existing bare UML diagrams and the dual exporting of diagrams enriched with ArchiTRIO features.

This paper presents the state of the art of the ArchiTRIO language and tool, and is structured as follows. Section 2 introduces a simple but significant example of real-time embedded system from the automotive domain (a semi-automatic gearbox controller). Section 3 presents the core of the tool supporting the ArchiTRIO language, with particular focus on the key design choices aimed at facilitating its interaction with existing and future UML 2.0 tools. Finally, Section 4 draws the conclusions and outlines some future work in the development of the tool.

2. THE ARCHITRIO APPROACH

In the ArchiTRIO approach to system modeling, the designer starts by drawing a class diagram representing the elements of the system. In addition, the precise internal structure of every composite class is described through an appropriate (UML 2.0) composite structure diagram. A particular composite element is the system itself, which is also represented by a UML composite structure diagram. It is at this point that the system designer can introduce ArchiTRIO logic formulas, to describe constraints (especially those on the temporal behavior of the system components) or desired properties that must be enforced by the design.

Let us illustrate the ArchiTRIO approach through a simple, although feature-wise fairly complete example. The system to be modeled is a semi-automatic gearbox for a sports car in which there are six speeds (plus reverse and neutral), and the user can decide to move one speed up/down with respect to the current one (it is of course impossible to move up from the sixth speed and down from the first one). The system is semi-intelligent, in that it does not allow changing to the lower gear if the current revolutions-per-minute (RPMs) are too high (which could damage the gearbox), nor switching to reverse while the car is moving forward. One requirement that the system must satisfy is that speeds are changed only if the user elected to do so.

Figure 1 shows the part of the UML class diagram depicting the interfaces of the elements of the system. Interface `GearBoxControlsI` contains the operations to change the gears (one up, one down, put in neutral, put in reverse); interface `GearBoxI` exports an attribute representing the current gear, interfaces `RPMSensorI` and `SpeedSensorI` also export an attribute each, which keep track of the current RPMs and speed of the car. In addition, there is a port, `SensorsInP`, which requires both sensor interfaces, and which is used to read data from the car sensors.

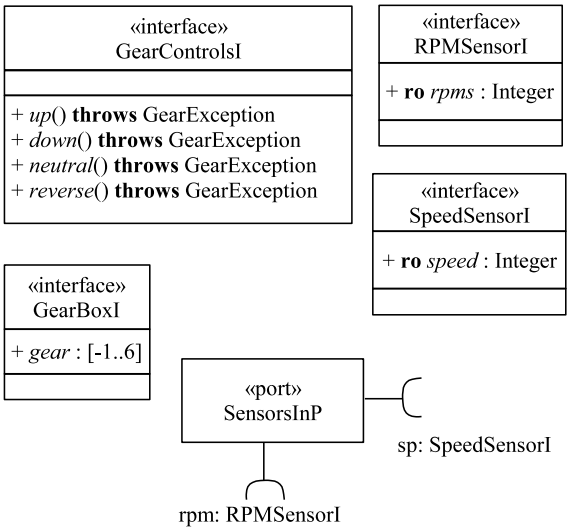

Figure 1: Interfaces of the gearbox components.

The interfaces and ports of Figure 1 are supported by the classes shown in the fragment of UML class diagram of Figure 2. The core of the system is the `Controller`, which offers an interface to allow the user (through the `Console`) to input the gear commands, and controls the gearbox through interface `GearBoxI`. In addition, `Controller` can read data from the car sensors `SpeedSensor` and `RPMSensor` through port `sens`, as shown below.

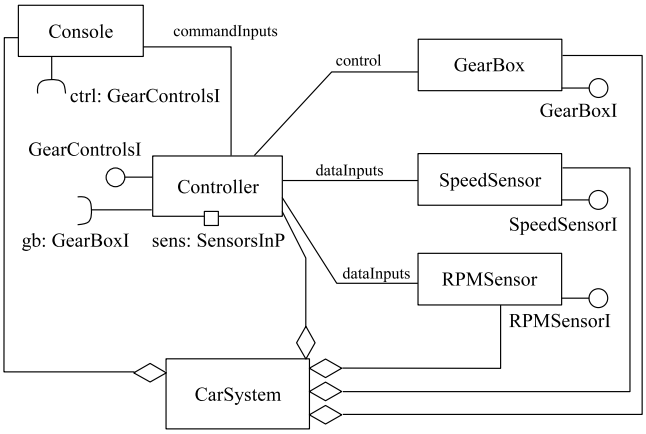

Figure 2: System elements.

The precise structure of the system (represented by class `CarSystem`) is described by the UML 2.0 composite structure diagram of Figure 3. Class `CarSystem` is composed of an instance each of classes `Console`, `Controller`, `GearBox`, `SpeedSensor` and `RPMSensor`. The interface `GearControlsI` required by the `csl` component of type `Console` is provided by the `Controller` object `ctrl`, which in turn uses interface `GearBoxI` from component `gbox`. In addition, the two interfaces `SpeedSensorI` and `RPMSensorI` required by port `sens` of object `ctrl` are provided by components `ss` and `rs`, respectively.

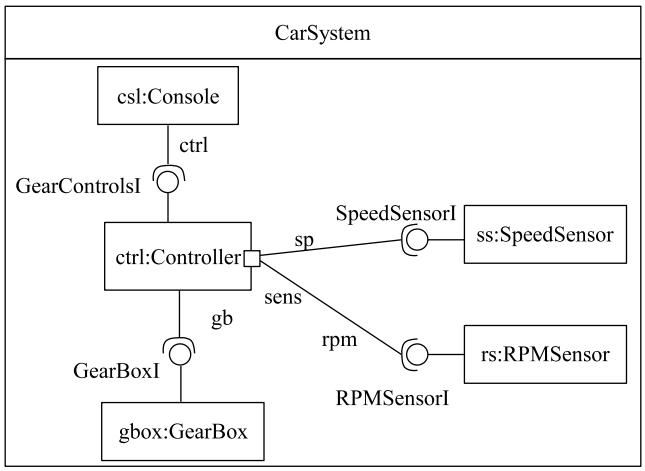

Figure 3: Composition of the car system.

After the structure of a class has been defined through a composite structure diagram, ArchiTRIO-specific elements [13, 12] may be added, if necessary. In particular, logic formulas can be introduced in classes to formally describe their dynamics (e.g. timing constraints), the assumptions they make on the dynamics of the objects with which they

interact, and the properties they are required to satisfy. At its core, ArchiTRIO is a higher-order temporal logic [12], which offers various features to represent in formulas the elements of a UML diagram, and a wide range of temporal operators taken from the TRIO temporal logic [1].

Let us focus on class `Controller` of Figure 2, in particular on the mechanism through which the car shifts down one gear. The first property that we would like to formalize is an upper bound on the duration of the downshifting operation: more precisely, that downshifting cannot take more than `T` time units (with `T` a system-dependent constant). In other words, we would like to state that no more than `T` time units can elapse from the moment an invocation of operation down of interface `GearControlsI` is received by the controller until the invocation terminates. This is captured by the ArchiTRIO axiom `gear_up_ax1`, which is added to class `Controller` through a separated, specific compartment as shown in Figure 4.

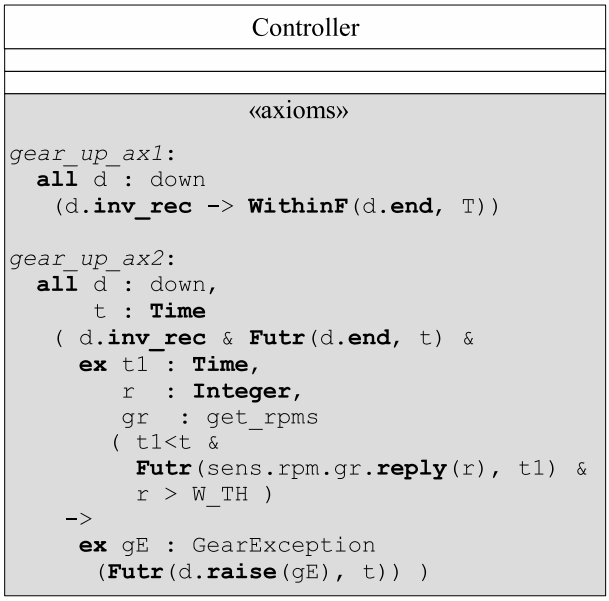

Figure 4: Fragment of the `Controller` class.

In axiom `gear_up_ax1`, `inv_rec` (resp. `end`) is a built-in logic predicate that represents the instant in which the invocation (`d` in this case) of an operation (`down`) is received by an object (resp. terminates). Formula `WithinF(d.end, T)` holds if and only if `d.end` occurs no more that `T` time units after the current instant (which is left implicit, as in TRIO [1]). Also, by definition, ArchiTRIO formulas are implicitly temporally closed with the `Alw` (always) operator [1], that is, they are true in every instant.

Notice that an operation can terminate in one of two ways: either successfully (without errors), or unsuccessfully (in which case an exception is raised). Predicate end represents when the invocation of an operation terminates, independently of its outcome.

Let us now analyze a second aspect of the downshifting mechanism: if the user (through the `Console` object) requests a downshift when the RPMs of the engine (as modeled by attribute `rpms` of the RPM sensor) are over a certain threshold `W_TH`, the request is rejected and an exception of type `GearException` is raised, since a downshift would further increase the RPMs, thus putting additional strain on the engine. This is expressed by axiom `gear_up_ax2` of Figure 4. `Futr` is a temporal operator similar to `Dist`, except that it requires the time distance (`t`) to be *positive*, while `reply` (resp. `raise`) is a built-in predicate that represents the instant in which the invocation of an operation terminates successfully (resp. with an exception). In addition, `reply(r)` (resp. `raise(gE)`) is a shortcut for stating that when the invocation ends, the returned value (resp. exception raised) is `r` (resp. `gE`). Finally, `get_rpms` is a built-in operation that is automatically introduced in ArchiTRIO when attribute `rpms` is defined, and returns the current value of the attribute.

The axioms of the objects composing the system should guarantee a number of properties of interest of the system. These properties can be captured through ArchiTRIO *theorems*, i.e. formulas that should be formally provable from the components' axioms. For example, a global property of the system of Figure 3 could be that a gear cannot be changed to the upper one unless the user issues the appropriate command. Such a property is expressed by theorem `gear_up_CN` of Figure 5. In this formula, `invoke` is a built-in predicate that represents when the invocation of an operation is issued by the client, while `sg.inv_rec(g1)` is a shortcut for expressing that the invocation `sg` of operation `set_gear` is received and has `g1` as a parameter. Also, `UpToNow(gbox.gear.value = g1-1)` is true if and only if there is a past interval ending in the current instant, in which the value of attribute `gear` was `g1-1`, while `WithinP` is the past-time counterpart of the `WithinF` operator used above.

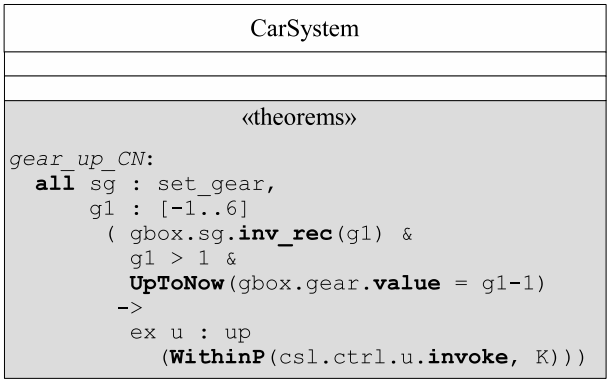

Figure 5: Fragment of the `CarSystem` class.

All the ArchiTRIO elements presented in this Section have been given a formal semantics, which can be used to carry out formal analysis of the system architecture (e.g. a mathematical proof of theorem `gear_up_CN` of Figure 5). However, it is not the goal of this paper to present the formal semantics of ArchiTRIO; the interested reader can refer to [12].

Let us conclude this section with a short note of comparison between ArchiTRIO and OCL [9] (the interested reader can find further comments in [13] and [12]). While there are some similarities between the two languages (ArchiTRIO axioms are essentially *constraints* on the dynamics of UML/ArchiTRIO classes, as Figures 4–5 could suggest), they are very different in scope and expressive power. They

differ in expressive power, since OCL does not include a definition of time. They differ in scope, in that OCL constraints express only operation pre/post-conditions and class invariants, while ArchiTRIO formulas can express a wider range of properties, such as complex interactions among system components. In fact, none of the formulas of Figures 4–5 can be expressed in OCL, and even if one took into account temporal extensions such as the one proposed in [4], still it would be impossible to formalize properties such as `gear_up_ax1` and `gear_up_CN`.

3. ARCHITRIDENT

The tool for the creation and management of ArchiTRIO models is part of the TRIDENT tool suite. TRIDENT is an environment for the development and analysis of time-critical systems based on the TRIO formal language. It is implemented on the Eclipse platform [5], and is being developed jointly by Politecnico di Milano and CEFRIEL.

As typical with Eclipse-based tools, TRIDENT is plugin-based, so it is by itself an open and evolving product. The environment is still in a prototypical stage, so many of the intended features are still incomplete. Figure 6 depicts both the original TRIO-based tools (on the right side), and the HOT-based tools (on the left side). HOT [6] stands for Higher-Order TRIO, the extension of TRIO on top of which ArchiTRIO is defined [12].

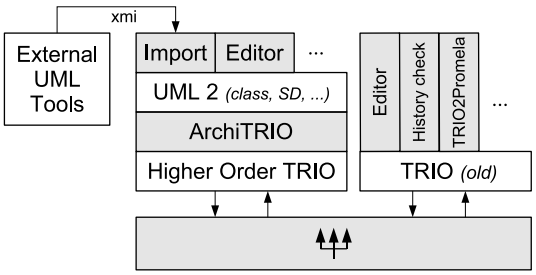

Figure 6: Overview of the TRIDENT environment.

Some of the most notable present features of the TRIO-based tools are the ability of editing complex TRIO specifications and *histories* (i.e. execution traces that may be used as test cases), and check their mutual compatibility. More recently, a plugin for supporting model-checking of TRIO specifications has been implemented [11].

As far as ArchiTRIO is concerned, currently there is an advanced-stage prototypical plugin (called ArchiTRIDENT), which is the focus of the present paper. This plugin is ideally based upon three main layers: 1) a subset of UML 2.0 diagrams (at present class and composite structure diagrams); 2) the ArchiTRIO layer, which provides semantics to 1) and possibly defines additional complex properties and behaviors; 3) the HOT layer, not yet implemented, which defines the semantics of 2).

Following the spirit of Eclipse, ArchiTRIDENT is designed for extensibility, so that the resulting tool is an integration of specialized components, working on a common data model. The plugin is characterized by a high level of modularity, easily supporting the evolution of the components and the integration with external applications (in particular UML 2.0-compliant editors and formal verification tools).

In the rest of this section we present the structure of the ArchiTRIDENT plugin and the elements of which it consists. First, we give an overview of the tool; then, we introduce the metamodel on which ArchiTRIDENT is founded, and the modules managing it; finally, we discuss the visual tools facilitating model creation and modification.

3.1 Architecture

The ArchiTRIDENT plugin is structured as a set of components, each of which is used to support a particular phase of the modeling process and management activities on specifications.

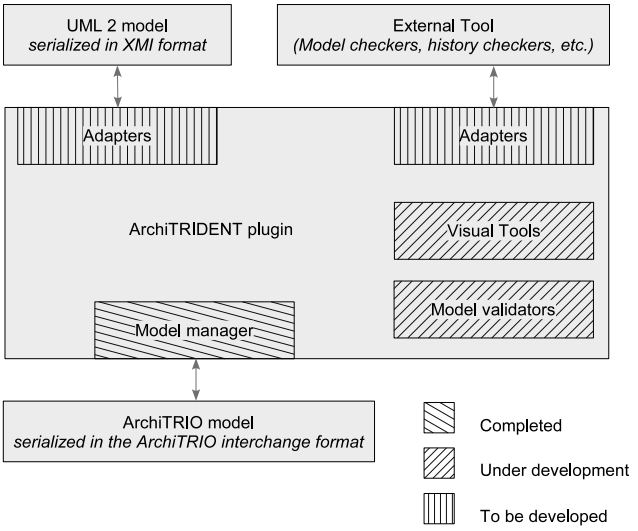

Figure 7: The components of the ArchiTRIDENT plugin.

The modules that are part of the ArchiTRIDENT plugin are sketched in Figure 7. At the core of the tool is a manager of ArchiTRIO models, which are serialized in the ArchiTRIO interchange format. The model manager supports the creation and modification of models and interacts with other components providing editing, viewing and model validation capabilities. In addition, the plugin is designed to include adapters for the integration with external verification tools and with UML 2.0-compatible applications. In particular, the ArchiTRIDENT plugin can be expanded with an array of adapters for the import/export of UML diagrams from/to external UML editors.

3.2 Metamodel

ArchiTRIDENT is conceptually built around the ArchiTRIO metamodel. Since ArchiTRIO shares concepts with UML 2.0 [12], its metamodel is essentially an extension of a subset of the UML one, in which ArchiTRIO-specific elements (such as axioms, theorems, etc.) are introduced.

The ArchiTRIO metamodel is defined through an XML Schema [3], to facilitate future integrations with external UML 2.0 tools through XMI-based adapters. This choice appears quite convenient, for both XML Schema provides support for model validation through the definition of the constraints a model must meet, and a number of tools are

available off-the-shelf to carry out the validation task.

As mentioned above, one of the advantages of having an XML-based interchange format is that it allows for better integration with XMI-supporting external tools. An extended XMI format can be easily defined adding ArchiTRIO-specific tags to existing XMI [8] documents (while keeping the rest of the structure of the documents intact), much like we do to ordinary UML classes descriptions (as in Figure 4). These enriched XMI descriptions may be managed by both pure UML tools, and ArchiTRIO-enabled tools (with only the latter taking into account all the tags). Because of the similarities in content and structure of the two formats, ArchiTRIO serialized models can be easily translated into extended XMI documents and vice-versa. A pre-requisite of this vision is the existence of an XMI standard widely supported by UML tools. Unfortunately, such is not the case at present, and the UML tools that support (often only partially) the 2.0 version of the language do not export in standard XMI format, but, rather, in proprietary, often incomplete variations. We plan to develop adapters for interaction and diagram import/export with tools that export in a standard, fully UML 2.0-compatible XMI format when these will be available.

The core Java classes representing the elements of the metamodel in the ArchiTRIDENT plugin are automatically generated from the Schema describing the ArchiTRIO metamodel, through a template-based code-generation approach. The code generation tool (XMLSpy by Altova [7]) automatically produces the code for parsing/unparsing the serialized models accordingly to the structure defined by the Schema and the custom code defined by the Java source code templates written in SPL (Spy Programming Language [7]). A Java class has been defined for each element of the metamodel. Each class provides methods to access the properties of an element instance.

Parsing the XMI serialized model generates a Document Object Model (DOM, [2]) tree that can be directly mapped to the XML document, and a hierarchical structure storing the elements of the model represented as Java objects, where every object corresponds to a node of the tree.

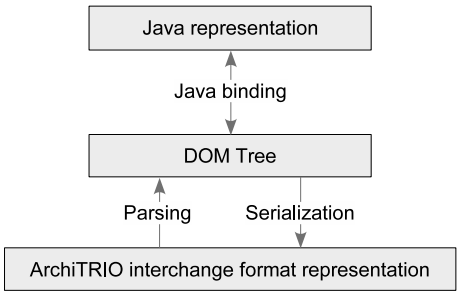

Figure 8: ArchiTRIO model structure.

Then, an ArchiTRIO model is essentially composed of the three levels shown in Figure 8. The XML document is at the first level; a representation of the DOM tree corresponding to the previously introduced XML document is at the second level; finally, the third level consists of a hierarchical structure of instances of Java classes representing the elements of the ArchiTRIO model.

ArchiTRIDENT provides features (parsers and inspectors), to check the consistency of a model (an important activity often overlooked in commercial UML tools). The validation process is composed of two main phases. During the first one a parser verifies that the serialized ArchiTRIO XML model is well formed and valid against the ArchiTRIO metamodel XML Schema. In this way structural errors and some constraint violations can be found. During the second phase of the validation process, dedicated inspectors check the well-formedness of the various elements defined in the model, such as ArchiTRIO formulas. A different checker is possibly defined for each kind of element of the metamodel.

3.3 Visual tools

The plugin provides multi-page editors and wizards to allow users to create and manipulate models and browse their different parts. For example, Figure 9 shows part of the Controller class of the car system presented in Figures 1–5.

Visual editors offer three different views on the elements of the model. The first one gives a schematic representation of the element structure, while the other two show a detailed textual serialization of the current element in ArchiTRIO XML interchange format and in ArchiTRIO source code format [13], respectively.

ArchiTRIDENT offers a set of wizards, which add creation functionalities to the multi-page editors. For example, Figure 9 shows the use of a wizard for the creation of axiom `gear_up_ax1` of class `Controller`.

ArchiTRIDENT's visual tools (such as the wizards) interact with core module to ensure the consistence of the model under construction.

4. DEVELOPMENT STATUS AND FUTURE WORKS

The ArchiTRIDENT plugin supports the ArchiTRIO modeling process, from the creation of the interfaces/classes to the introduction of the axioms describing the objects' behavior. The development has focused so far on the definition of the structure of the plugin and of the extension mechanisms for the integration of external tools.

While the core of ArchiTRIDENT has reached a good level of stability, parts of the tool have yet to be completed, in particular the model validation modules.

At present, ArchiTRIDENT lacks a graphical editor, one that would allow designers to create the diagrams of Figures 1–3. Such an editor, though, would be unnecessary if reliable (commercial or otherwise) UML tools fully compliant with the 2.0 standard (and capable of exporting in a fully-standard XMI format) were available. In fact, we plan to develop adapters for communication with such tools through XMI files when these will be available. We envision a scenario in which a designer uses a UML 2.0 tool to define the structural aspects of a model (i.e. Figures 1–3), and then passes to the ArchiTRIDENT plugin to add dynamic constraints and to check the model's validity (Figures 4–5).

Finally, we plan to validate our approach by applying it to industrial-strength case studies from two classes of systems: embedded systems (with particular reference to the automotive domain) and Flexible Manufacturing Systems. In both cases, the target users will be system engineers familiar with the UML notation, but with little or no experience in the use of formal methods: we will work with them to create fully formal ArchiTRIO models from their initial, semi-formal UML diagrams.

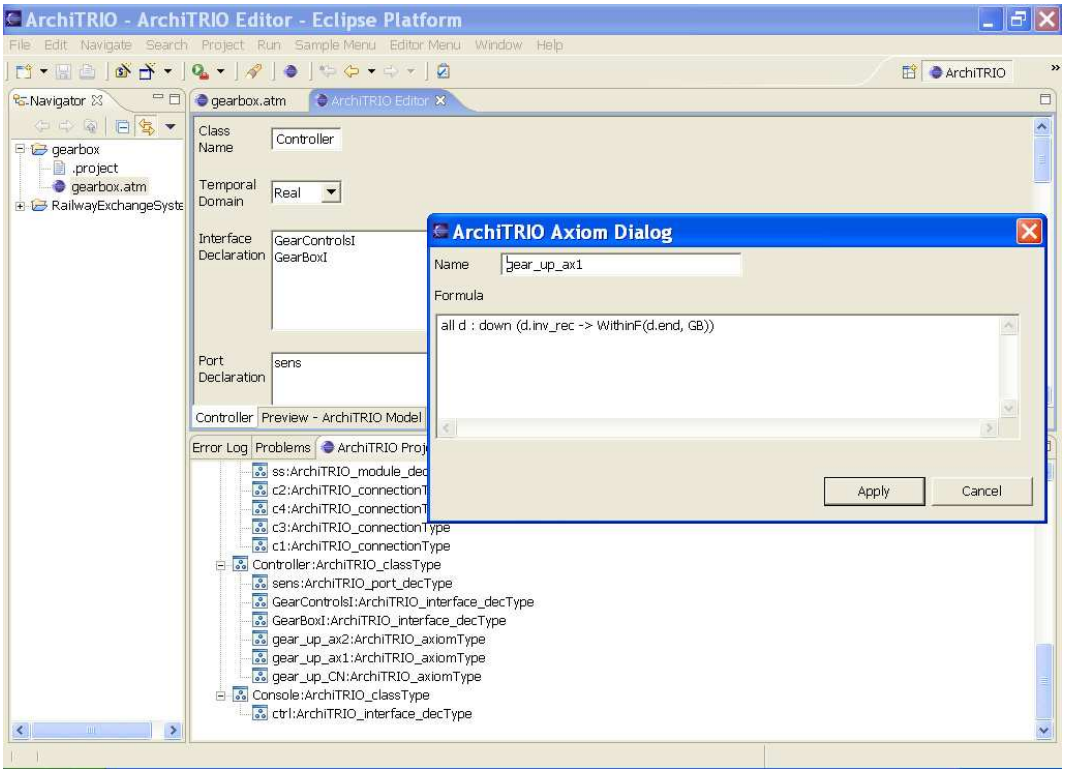

Figure 9: ArchiTRIDENT view of class `Controller` of Figure 4.

5. ACKNOWLEDGMENTS

The authors would like to thank Dino Mandrioli for his contribution in defining the ArchiTRIO language.

6. ADDITIONAL AUTHORS

Giordano Sassaroli (CEFRIEL, Via Fucini 2, 20133 Milan, Italy, email: `sassarol@cefriel.it`).

7. REFERENCES

[1] E. Ciapessoni, A. Coen-Porisini, E. Crivelli, D. Mandrioli, P. Mirandola, and A. Morzenti. From formal models to formally-based methods: an industrial experience. *ACM TOSEM*, 8(1):79–113, 1999.

[2] The World Wide Web Consortium. Document object model level 3 core. W3c recommendation, W3C, 2004.

[3] The World Wide Web Consortium. Xml schema part 1: Structures. W3c recommendation, W3C, 2004.

[4] S. Flake and W. Mueller. Past- and future-oriented time-bounded temporal properties with OCL. In *Proc. of the 2nd Int. Conf. on Software Engineering and Formal Methods*, pages 154–163, 2004.

[5] Eclipse Foundation. `http://www.eclipse.org`.

[6] C. A. Furia, D. Mandrioli, A. Morzenti, M. Pradella, M. Rossi, and P. San Pietro. Higher-order TRIO. Technical report, DEI, Politecnico di Milano, 2004.

[7] Altova Gmbh. XMLSpy user manual and programmers' reference. Technical report, Altova Gmbh, 2005.

[8] Object Management Group. UML 2.0 diagram interchange specification. Technical report, OMG, 2003. ptc/03-09-01.

[9] Object Management Group. UML 2.0 OCL specification. Technical report, OMG, 2003. ptc/03-10-14.

[10] Object Management Group. UML 2.0 superstructure specification. Technical report, OMG, 2003. ptc/03-08-02.

[11] A. Morzenti, M. Pradella, P. San Pietro, and P. Spoletini. Model-checking TRIO specifications in SPIN. In *Proc. of the 12th Int. Symp. on Formal Methods*, volume 2805 of *LNCS*, pages 542–561, 2003.

[12] M. Pradella, M. Rossi, and D. Mandrioli. ArchiTRIO: A UML-compatible language for architectural description and its formal semantics. In *Proc. of FORTE 2005: 25th IFIP WG 6.1 Int. Conference*, volume 3731 of *LNCS*, pages 381–395, 2005.

[13] M. Pradella, M. Rossi, and D. Mandrioli. A UML-compatible formal language for system architecture description. In *Proc. of the 12th SDL Forum*, volume 3530 of *LNCS*, pages 234–246, 2005.

[14] H. Saiedian, J. P. Bowen, R. W. Butler, D. L. Dill, R. L. Glass, D. Gries, A. Hall, M. G. Hinchey, C. M. Holloway, D. Jackson, C. B. Jones, M. J. Luts, D. L. Parnas, J. Rushby, J. Wing, and P. Zave. An invitation to formal methods. *IEEE Computer*, 29(4):16–30, 1996.