

CHARMY: An Extensible Tool for Architectural Analysis

Paola Inverardi, Henry Muccini and Patrizio Pelliccione
Dipartimento di Informatica
University of L'Aquila
Via Vetoio - L'Aquila, Italy
[inverard,muccini,pellicci]@di.univaq.it

ABSTRACT

CHARMY is a framework for designing and validating architectural specifications. In the early stages of the software development process, the CHARMY framework assists the software architect in the design and validation phases. To increase its usability in an industrial context, the tool allows the use of UML-like notations to graphically design the system. Once the design is done, a formal prototype is automatically created for simulation and analysis purposes. The framework provides extensibility mechanisms to enable the introduction of new design and analysis features.

Categories and Subject Descriptors

D.2.11 [Software Architectures]; I.6.4 [Model Validation and Analysis]

General Terms

Design, Verification

Keywords

Model checking, Software Architectures

1. INTRODUCTION

Nowadays industries are increasing their interests in *analyzing and validating architectural choices*, both behavioral and quantitative. Software Architecture (SA)-based analysis methods have been introduced to provide several value-added benefits, such as system deadlock detection, performance analysis, component validation and much more [9]. Despite the flourishing of research work on architectural analysis, very few tools have been proposed to support SA-level analysis, and many of them are not anymore supported or difficult to be introduced in an industrial context. Thus, *how to automate the SA-based analysis process in a way useful for current industrial needs* is a topic which requires a careful investigation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ESEC-FSE'05, September 5–9, 2005, Lisbon, Portugal.
Copyright 2005 ACM 1-59593-014-0/05/0009 ...\$5.00.

This paper introduces CHARMY, a tool which allows the specification of a software system SA through diagrammatic, UML-based notations, and *the validation of the architectural specification conformance with respect to certain functional requirements*. CHARMY offers a graphical user interface to draw state diagrams and scenarios, used to specify the SA behavior and the functional requirements, respectively. A translation engine automatically derives formal specifications out of the diagrammatic notations, and the SPIN [11] model-checker is used for automatic verification on such specifications. XMI is the output format of CHARMY. Moreover, the framework provides extensibility mechanisms (via a plugin-based architecture) to enable the introduction of new features and to help the integration with other existing analysis tools.

The tool main benefits are that it is UML-based (thus easily integrable in industrial development processes), it automatically produces a formal prototype of the SA and model-checks it with SPIN (without requiring formal languages skills), and it is extensible, due to its plugin architecture.

2. CHARMY FEATURES

CHARMY allows the specification of a software architecture by means of both a topological (static) description and a behavioral (dynamic) one [10]. To increase the acceptability of our tool in industrial contexts we use a UML-based notation (stereotyped class diagrams for the topology and state diagrams for the behavior). The tool has been used in real case studies both industrial and academic as summarized in Section 6.

CHARMY allows the specification of the SA topology in terms of components, connectors and relationships among them, where components represent abstract computational subsystems and connectors formalize the interactions among components. The internal behavior of each component is specified in terms of state machines.

The CHARMY tool performs several checks, at the SA specification level, in order to find static specification errors: a) in a state diagram it is not possible to introduce two states with the same name; b) each state diagram must contain one and only one initial state; c) for each send (receive) message in a component, there must exist a receive (send) message in another component; d) sequence diagrams can contain only messages already present into the state diagrams of the involved components; e) the sender and the receiver of a message must be the same (components) in the sequence diagrams and in the state diagrams; f) messages with the same name must have the same number of parameters.

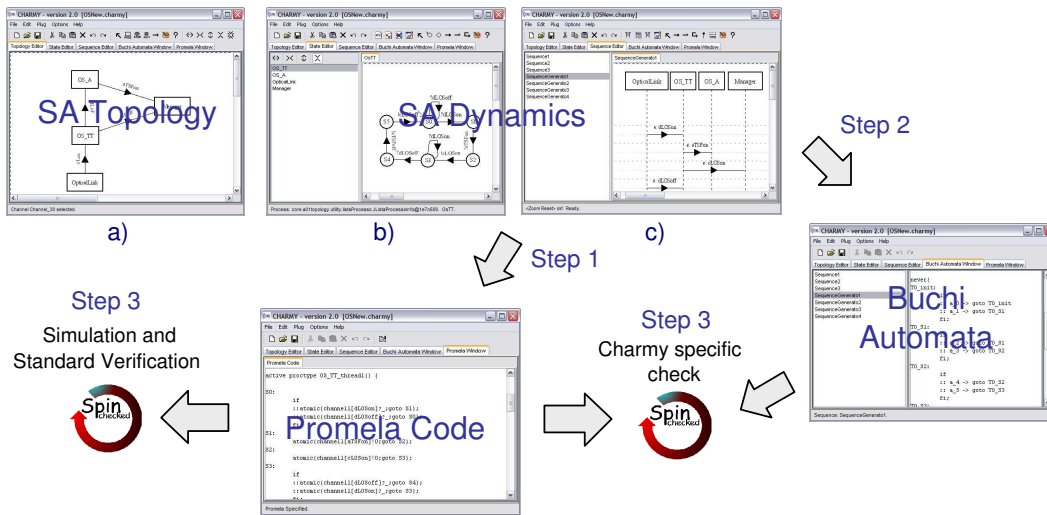


Figure 1: The CHARMY Tool

Once the SA specification is available, the translation feature is used to obtain from the model-based SA specification, a formal executable prototype in Promela (the specification language of SPIN) [11]. On the generated Promela code, we can use the SPIN standard features to find, for example, deadlocks or parts of states machines that are unreachable.

Temporal properties are modeled using Property Sequence Charts PSC, sequence diagram notation used to specify properties [3]. Each sequence diagram represents a desired behavioral property we want to check in the Promela (architectural) prototype. The PSC2BA algorithm automatically translates PSC into Büchi automata (the automata representation for LTL formulae) while the SPIN model-checker is used to validate the temporal properties on the Promela code.

Note that this translation process is fully automated.

The aforementioned features are implemented through a *graphical editor* component which allows the specification of both the topology and the behavior of the software architecture. This component is composed by the *topology editor* which allows the specification of the SA topology (Figure 1.a) and the *thread editor* which allows the specification of the internal behavior of each component (Figure 1.b). The *sequence editor* allows the draw of sequence diagram representing desired behavioral properties we want to check (Figure 1.c). A *translator utility* converts state diagrams into Promela (Figure 1.step1). The translation algorithm (Figure 1.step2), described in [3], allows an automatic translation of the sequence diagrams into the Büchi automata formalism, comprehensible by the model checking engine supported by SPIN. More details on the CHARMY features can be found in [12].

3. THE CHARMY PLUGIN SA

The CHARMY plugin Software Architecture is driven by requirements of easy extensibility of the initial core in several directions. Notably we want to be able to extend the *kind of analysis* we may perform on a software architecture and we want CHARMY to be *easily integrated with other existing tools*.

The CHARMY tool architecture is shown in Figure 2. Taking a look at the CHARMY Core macro-component, it is composed by the **Data Structure** component, the **Plugin Manager** which allows the handling of the plug of a new component in the core system, the **GUI** which receives stimuli by the users, and activates the **Action Manager** and the **Event Handler**.

The **Core Plugin** meta-component contains a set of core plugs to edit the software architecture topology, the state machines and the scenarios respectively.

The **Standard Plugin** contains a set of standard plugs to implement the translation from sequence diagrams to Büchi automata and from state machines to Promela code. Moreover, this component will contain others future plugs.

The CHARMY core handles the plugin management by specifying: *i*) how a new plug should be implemented, *ii*) how the core system has to recognize the plug and use it, and *iii*) how the core and plug components should interact.

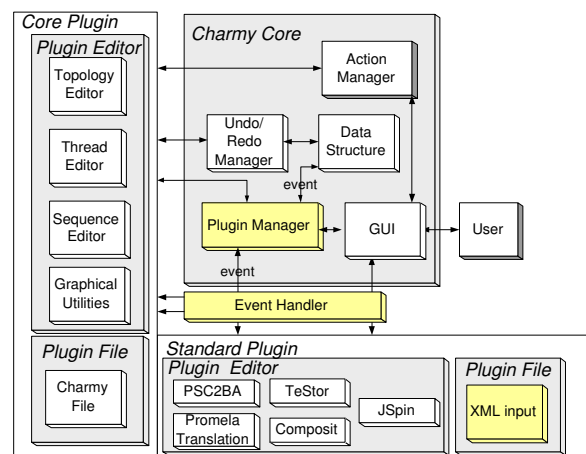


Figure 2: The CHARMY Plugin Architecture

Figure 3 graphically summarizes these aspects.
i) Implementing a new plug: when a new component needs to become a plugin, it has to implement two interfaces: the

“IMainTabPane” and the “IFilePlug”. The IMainTabPane interface handles the data information related to the windows. Here we have methods which allow the reception of information from the editor components (Topology, Sequence and Thread components). The IFilePlug interface, instead, needs to be implemented when the plug requires to save or open a file.

ii) Recognizing the new plug: when a new plug is created and wants to be inserted, the core system needs to be informed about this. The solution we adopted is to create an .xml file (called plugin.xml) which contains all information needed.

iii) Interaction: when a data is modified inside the core system, an event is sent by the Event Handler component to the plug. This event informs the plug of which kind of modification has been made over the data (e.g., insert, modify, delete) and sends a clone of the data itself to the plug. A plug, in order to receive the event, has to be registered as a listener of the event itself.

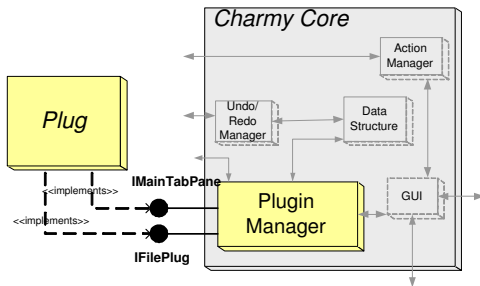


Figure 3: Plug and Core

4. THE IMPLEMENTED PLUGINS

In this section we briefly describe the CHARMY standard plugins implemented so far and illustrated in Figure 2.

4.1 PSC2BA: From PSC to Büchi Automata Plugin

The PSC2BA algorithm translates Property Sequence Charts PSC [3] into Büchi automata. A scenario editor is used to draw PSC, while the PSC2BA plugin translates them into Büchi automata. Such automata is used in Step 3 (see Figure 1) for CHARMY specific checks.

4.2 The Promela Translation Plugin

The Promela code generation plugin allows the translation of components’ state machines into Promela code. The generated Promela code is used for running SPIN standard features to find, for example, deadlocks or parts of states machines that are unreachable. The translation algorithm is described in [12] while the plugin is downloadable with CHARMY.

4.3 The TESTOR Plugin

TESTOR [15] (TEst Sequence generaTOR) is an algorithm for extracting test sequences from model-based specifications as produced by practitioners. A behavioral model of each component is provided in the form of UML state diagrams. Such models explicitly specify which components interact and how. Sequence diagrams (inSD) are used as

an abstract or even incomplete specification of what the test should include, by representing test directives. TESTOR takes in input UML state and sequence diagrams and synthesizes more detailed sequence diagrams (outSD) (conforming to the inSD) by recovering missing information from state diagrams. The output represents test sequences specified in the form of more informative scenarios.

The TESTOR algorithm has been implemented has a plugin component for CHARMY and a beta version is currently available [6]. More details on the TESTOR plugin may be found in [15, 6].

4.4 Compositional Analysis of Middleware - based SA Plugin

Summarizing the approach in [5], our aim is to model check middleware-based SA with respect to a subset of LTL system properties by means of Compositional Reasoning. Our approach exploits the structure imposed on the system, by the SA. The idea is to decompose the verification of a global property, into the verification of a number of properties that hold locally on the architectural components. The architectural structure helps in deriving the validity of the whole system from the validity of the local properties.

This plugin realizes and partially automatizes the theory presented in [5]. A graphical editor allows the design of the Software Architecture without detailing how the components interact. Thus the communication between each pair of components is represented with a simple link connection, distinguishing between invocation of a service and results retrieval. To obtain a refined SA, the software architect can select a defined middleware. The proxies’ models (used to bridge the application with the middleware) are automatically generated. More details on the compositional verification plugin may be found in [6].

4.5 JSpin Plugin

The rationale is to introduce in CHARMY a graphical interface for the SPIN model checker. To accomplish this task we exploit JSpin, a Java GUI for Spin developed by Moti Ben-Ari [13].

5. RELATED WORK

When dealing with tools for functional analysis of software architectures, we may distinguish between proposed tools, still supported tools and tools usable in industrial contexts.

In the first class, i.e. the proposed ones, we may list all of those ones (mainly academic) introduced in the ’90s to model and analyze specific Architecture Description Languages (ADLs) (e.g., Aesop, ArTek, C2, Darwin, LILEANNA, MetaH, Rapide, SADL, UniCon, Weaves, Wright) [14].

Currently only some ADLs seem to be still supported and in use. All such still-in-use tools are somehow easy to use, even if none of them makes use of UML-like notations. Their main limitation is that each of them focusses on a particular analysis technique, leaving other techniques unexplored. Moreover, each of them uses a different notation for SA specification, thus making any integration difficult.

6. SOME CONSIDERATIONS

CHARMY has been thought in order to be *easily integrated in industrial projects*: the model checker engine complexity is hidden, providing the software engineer an automated,

easy to use tool which takes in input the architectural models in a UML-based notation, creates the prototype and automatically analyzes the prototype reducing as much as possible human intervention.

CHARMY supports the software architect *in the design process in order to produce space efficient models*.

In fact, CHARMY provides guidelines on how to model the system and automatically generates an optimized Promela code, thus allowing an exhaustive analysis through model-checking. The experience shows that there is generally a considerable difference in efficiency and memory size between models developed by a “casual” user and models developed by an “expert” user. Then by using CHARMY, the usual problems of state explosion and model memory size are mitigated, without requiring particular knowledge to users.

CHARMY has been used in *several case studies* both industrial and academic: *NICE* a joint work with Marconi Mobile Lab. NMS C2 (L’Aquila-Italy) that operates in a naval communication environment [8]. *Siena* and COMETA a publish/subscribe middleware and its extension to handle mobility [4]. *Engineering Order Wire (EOW)* a joint work with Siemens C.N.X. S.p.A., R. & D. (L’Aquila-Italy). EOW is an application that supports a telephone link between multiple equipments by using dedicated voice link channels [2].

7. FUTURE WORK

Interesting extensions are planned to take place:

Verification Engine: since we are not tied to use the model checker SPIN, we are currently investigating the use of SMV or Bogor as model-checking engines. In the case of Bogor, it is very interesting to take advantage of its plugin structure in order to define a customized search algorithm for SA.

Time and Space savings techniques: recently much effort focusses in techniques that operate on the input of the model checker (models) in order to improve time and space efficiency: abstraction, symmetry and compositional reasoning [7] are the currently evaluated solutions. The plugin SA of CHARMY will allow the introduction of new features to handle these new techniques.

Architecture Description Language: the SA topology editor will be extended by following the representation provided by common architecture description languages and by taking into consideration our experience in SA-based modeling and analysis in industrial contexts. In particular, concepts and formalisms coming from both architecture description languages, UML and XML representation of SA [17, 1] will be taken into consideration. By using existing architectural languages, we plan also to be able to reuse existing dependence analysis [16] and architectural slicing [18] techniques, already automated by other tools.

8. REFERENCES

- [1] ADML: Architecture Description Markup Language. http://www.opengroup.org/architecture/adml/adml_home.htm, Last Modified: December 2002. Open Group.
- [2] A. Bucchiarone, H. Muccini, P. Pelliccione, and P. Pierini. Model-Checking plus Testing: from Software Architecture Analysis to Code Testing. In Proc. Int. Workshop on Integration of Testing Methodologies, ITM ’04. LNCS n.3236
- [3] M. Autili, P. Inverardi, P. Pelliccione. Graphical Scenarios for Specifying Temporal Properties: an Automatic Approach. Technical report, University of L’Aquila, April 2005.
- [4] M. Caporuscio, P. Inverardi, and P. Pelliccione. Formal analysis of architectural patterns. In *First European Workshop on Software Architecture - EWSA 2004*, 21-22 May 2004, St Andrews, Scotland, UK. LNCS n.3047.
- [5] M. Caporuscio, P. Inverardi, and P. Pelliccione. Compositional verification of middleware-based software architecture descriptions. In *Proceedings of the International Conference on Software Engineering (ICSE 2004)*, Edinburgh, 2004. IEEE Computer Science Press.
- [6] Charmy Project. Charmy web site. <http://www.di.univaq.it/charmym>, February 2004.
- [7] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2001.
- [8] D. Compare, P. Inverardi, P. Pelliccione, and A. Sebastiani. Integrating model-checking architectural analysis and validation in a real software life-cycle. In FME 2003, LNCS 2805, pages 114–132, Pisa, 2003.
- [9] Formal Methods for Software Architectures. Tutorial book on Software Architectures and Formal Methods. Eds. M. Bernardo and P. Inverardi, LNCS 2804, 2003.
- [10] D. Garlan. Software Architecture: a Roadmap. In A. Finkelstein (Ed.), *ACM ICSE 2000, The Future of Software Engineering*, pp. 91-101, 2000.
- [11] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, September 2003.
- [12] P. Pelliccione, P. Inverardi, and H. Muccini. Charmy: A framework for Designing and Validating Architectural Specifications. Submitted for publication. Technical report, University of L’Aquila, April 2005.
- [13] jSpin - A Java GUI for Spin. <http://stwww.weizmann.ac.il/g-cs/benari/jspin/>.
- [14] N. Medvidovic and R. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 2000, 26(1), pp. 70-93.
- [15] P. Pelliccione, H. Muccini, A. Bucchiarone, and F. Facchini. TeStor: Deriving Test Sequences from Model-based Specifications 8th International SIGSOFT Symposium on Component-based Software Engineering. May 2005, St. Louis, Missouri, USA. LNCS 3489, pp. 267-282.
- [16] J. A. Stafford, and A.L. Wolf, Architecture-level dependence analysis in support of software maintenance. In *Third International Software Architecture Workshop* (Orlando, Florida, November 1998), pp. 129–132.
- [17] xADL 2.0 Architecture Description Language. <http://www.isr.uci.edu/projects/xarchuci/>, 2005.
- [18] J. Zhao Software Architecture Slicing. In *Proceedings of the 14th Annual Conference of Japan Society for Software Science and Technology* (1997).