

**Title:** Static and Dynamic Modeling and Analysis of Software Architecture

**Authors:** Roshanak Roshandel, Nenad Medvidovic

Computer Science Department  
University of Southern California  
Los Angeles, CA 90089-0781, USA  
{roshande, neno}@usc.edu

**Contact:** Roshanak Roshandel  
(213) 740-6504  
Computer Science Department  
University of Southern California  
Los Angeles, CA 90089-0781, USA  
[roshande@usc.edu](mailto:roshande@usc.edu)

**Focus:** Architectural representation and analysis

# Static and Dynamic Modeling and Analysis of Software Architecture

## Introduction

One of the goals of software architecture research has been to equip developers with powerful representation and analysis tools, which enables them to effectively leverage high-level, formal architectural models in the engineering of large, complex software systems [11,14]. A number of architecture description languages (ADLs) have been proposed to provide just such modeling and analysis support [9]. They are usually characterized by formal syntax and semantics intended to facilitate the representation and analysis of *specific* aspects of a software system's architecture. Typically they are narrowly focused on the specification and analysis of such architectural aspects as deadlock analysis, simulation, static structure, and so forth [9]. The ADLs leverage those narrow foci to provide some sophisticated analysis capabilities. At the same time, the narrow focus ultimately hampers a given ADL's general applicability. For this reason, several approaches have recently attempted to provide mechanisms for bridging different ADLs and enabling their combined use in a software development project: ACME [4], xADL [1,6], and UML [7]. However, for the most part these approaches have focused on bridging the divergent syntactic bases of the ADLs, while the issue of ADLs' semantics has not been explicitly and completely addressed to date.

In general, the different ADLs' approaches to architectural specification could be categorized as *static* and *dynamic*. Static specification techniques enable representation of system structure and possibly other aspects, such as behavior, performance, and distribution profile. The models of these system aspects are static if they declaratively specify properties that must hold at *discrete* points during the system's life span, but do not provide details on *how* those properties are to be achieved. On the other hand, dynamic modeling enables an architect to provide a more *continuous* view of how to arrive at a given property or desired state during a system's life span.<sup>1</sup>

We have developed a framework to relate *both* the syntax and the semantics of two architecture modeling notations: C2SADEL and StateCharts. C2SADEL, an ADL for modeling C2 architecture in the C2 style, supports modeling of *static* semantics of an architecture (via declarative behavioral "snapshots" expressed in component invariants and operation pre- and post-conditions), while StateCharts supports modeling of a system's *dynamic* semantics (via a continuum of system states and state transitions, events triggering those transitions, and actions caused by those events). Using the two notations in tandem allows architects to select the notation with which they have had more extensive experience. Furthermore, the combination of C2SADEL and StateCharts expands the range of possible architectural analyses. Ultimately, this work will significantly enhance our already existing support for automatically generating partial implementations from architectural models [8]. Such capabilities are currently not widely provided by ADLs. This deficiency is particularly notable in designing today's complex, reactive, and mission critical systems.

## C2SADEL and StateCharts

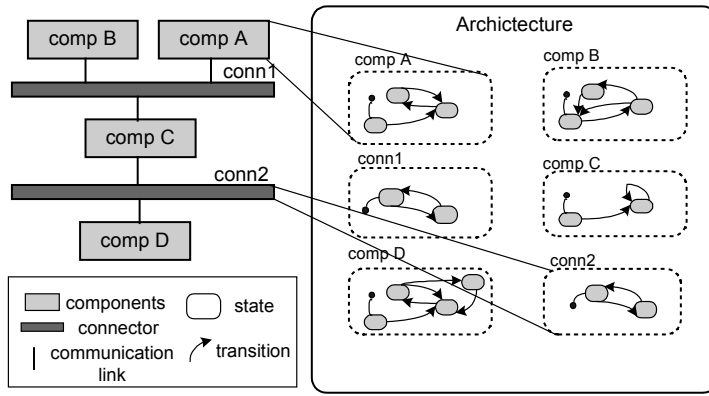
An architecture is specified in C2SADEL in three parts: component types, connector types, and topology. The topology, in turn, defines component and connector instances for a given system and their interconnections. Each C2SADEL component has a *name*, a set of *interface elements*, an associated *behavior*, and (possibly) an *implementation*. Each interface element has a direction indicator (*provided* or *required*), a name, a set of parameters, and (possibly) a result. Each parameter, in turn, has a name and a type. A component's behavior consists of an *invariant* and a set of *operations*. The invariant is used to specify properties that must be true of all component states. Each operation has pre-conditions, post-conditions, and (possibly) a result. Like interface elements, operations can be *provided* or *required*. Only provided operations will have an implementation in a given component. The pre- and post-conditions of required operations express their *expected* semantics. An interface element and its corresponding operation comprise a *component service*.

As initially proposed by Harel, StateCharts extend the conventional formalism of finite-state machines with concepts such as hierarchy and concurrency [5]. StateCharts are capable of describing dynamic component behavior based on the internal and external stimuli that transform a component's current state to a new state. StateCharts model systems using their *states* and the *transitions* among the states. A state machine is an abstract machine that defines a set of conditions of existence named *states*, a set of *events* that cause pre-defined changes in states, and a set of behaviors or *actions* performed in each of these states as a result of the occurrence of events. Simple and composite states, transitions, events, actions, guards on events, and state entry variables are among other state machine elements that we have used to build the bridge between the StateCharts and C2SADEL models.

We argued above for coupling static and dynamic modeling techniques to expand the scope of architectural analysis. Static architectural specification such as that produced by C2SADEL is essential to design the topology of the system, separate concerns by creating components and connectors, and specify their behavioral "snapshots" at well-defined points in the modeled system's execution. However, static semantics are incapable of sufficiently

---

<sup>1</sup> In the context of this research, "dynamism" refers to the run-time behavioral semantics of a given architecture, rather than the dynamic reconfiguring of the architectural topology.



**Figure 1. Example architecture in C2 style and in StateCharts**

executing state machines, each of which describes components and connectors within the architecture. The state machines of components and connectors interact through executions of events and actions. These event and action executions act as synchronizers among state machines. Figure 1 visualizes the notion of an architecture in both the C2SADEL and StateCharts. In the interest of clarity, labels on the transitions, events, actions, parameters, and conditions have been omitted [13].

C2SADEL specification of a component utilizes concepts such as state-variables, invariants, interfaces, operations, and pre- and post-conditions. Our mapping between these elements and those of StateCharts are shown in Table 1. These rules can be used to aid the creation of StateCharts that adhere to the properties specified in a C2SADEL model of the system. Once we have the state-based model that is intended to extend the original component-connector based model, we have access to significant analysis capabilities that will help us in designing the system more accurately and making the transition from architecture to implementation smoother. StateCharts developed based on our approach may serve in two different ways: they may be used as a specification tool for component behavior and can help in generating more complete code from the specification; they may also be used as a testing mechanism to ensure correct run-time system behavior with regard to the specification.

## Tool Support

To automatically establish the consistency of the C2SADEL and StateCharts models, we have developed a prototype tool. This tool is an integration of Argus-I, an OTS environment for the creation and analysis of StateCharts [2,3], with DRADEL, an environment for supporting architecture-based development using C2SADEL [8]. Argus-I's internal model supported architectural modeling at the level of component interfaces but did not provide any support for modeling and analysis of component behavior. We modified Argus-I's model by providing support for behavioral modeling. This enables us to leverage Argus-I's structural and behavioral analysis, from type checking to model checking and simulation. In addition, DRADEL enables us to analyze a C2SADEL specification to ensure conformance to the C2-style syntactic and semantic rules. The integrated tool is capable of checking the consistency of the two models in addition to static and dynamic analyses mentioned. The process of checking the consistency of the models is two fold. First, we use Argus-I to create a StateCharts model and analyze it for syntactic correctness. This includes checking for non-determinism, conflicting states, and conflicting transitions, as well as making sure all transitions are properly attached to the corresponding states. We have augmented Argus-I with a component that to create an internal representation of the StateCharts model of the system. This internal representation can be directly exported into DRADEL. Separately,

describing component behavior at run-time. We have used state machines with their hierarchical and concurrent properties to describe the behavior of components and connectors in the system; however, the topology of those components and connectors cannot be explicitly specified in StateCharts but must be specified in C2SADEL.

We have developed a set of rules to be used as a guideline for creating StateCharts that describe the behavior of components and connectors based on their C2 specification. We model an architecture in StateCharts as a composite state machine. This state machine contains other concurrently

**Table 1. Mapping between C2SADEL and StateCharts elements**

C2SADEL	StateCharts
Architecture	Composite state machine
Components	State machine
Connectors	State machine
State variables	State's variable
Invariants	Implicit in designing the states
Required Interface/Operation	Events
Provided Interface/Operation	Actions or events
Interface/Operation parameters	Parameters on transitions
Pre-conditions	Guards
Post-conditions	Modified state entry variable

DRADEL parses and analyzes a C2SADEL specification of the same system. The two models are then analyzed against one another using DRADEL's consistency checking facility and any mismatches between them are identified. The architecture of the integrated environment is shown in Figure 2.

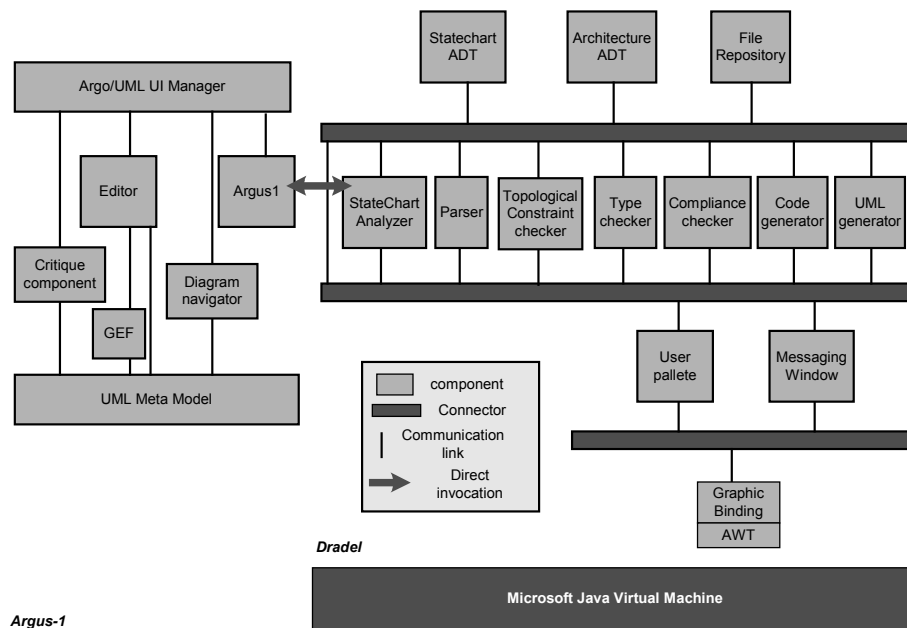


Figure 2. Integrated environment's architecture

## References

1. Dashofy E., van der Hoek A., and Taylor R.N., A Highly-Extensible, XML-Based Architecture Description Language. In *Proceedings of The Working IEEE/IFIP Conference on Software Architecture*, Amsterdam, The Netherlands, August 2001.
2. Dias, M., Vieira, M., Richardson, D. J., Analyzing Software architecture with Argus I. ICSE2000 - Formal Research Demo, June 2000, Limerick, Ireland.
3. Dias, M., Vieira, M., Software Architecture Analysis based on Statechart Semantics. *International Workshop on Software Specification and Design, IWSSD-10*, San Diego, CA, USA, November 2000.
4. Garlan D., Monroe R., and Wile D., ACME: An Architecture Description Interchange Language, in *Proceedings of CASCON'97*, November 1997.
5. Harel, D. Statecharts: A Visual Formalism for Complex Systems, *Science of Computer Programming*, 8, 231-274 , 1987.
6. Khare R., Guntersdorfer M., Oreizy P., Medvidovic N., and Taylor R.N., xADL: Enabling Architecture-Centric Tool Integration with XML. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, January 2001.
7. Medvidovic N., Rosenblum D.S., Robbins J.E., and Redmiles D.F., Modeling Software Architectures in the Unified Modeling Language. To appear in *ACM Transactions on Software Engineering and Methodology*, 2001.
8. Medvidovic N., Rosenblum D.S., and Taylor R.N., A Language and Environment for Architecture-Based Software Development and Evolution, in *Proceedings of the 1999 International Conference on Software Engineering*, pp. 44-53, 1999.
9. Medvidovic N., and Taylor R.N., A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering* 26(1), pp. 70-93, January 2000.
10. Object Management Group, Unified Modeling Language specification, v1.3 spec, June 1999 (<http://www.rational.com/uml/resources/documentation/index.jsp>)
11. Perry D.E., and Wolf A.L., Foundations for the Study of Software Architectures. *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40-52, October 1992.
12. Robbins J, Argo/UML: Cognitive Support for Object-Oriented Design, version 0.6.2, April 1999 (<http://www.ics.uci.edu/pub/arch/uml/>)
13. Roshandel R., and Medvidovic N., Coupling Static and Dynamic Semantics in an Architecture Description Language. To appear in *Proceedings of the Working Conference in Complex and Dynamic System Architecture, CDSA 2001*, Brisbane, Australia, December 2001.
14. Shaw M., and Garlan, D., *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996