Scenario-Driven Analysis of Component-Based Software Architecture Models

Prasanta Bose
{bose@isse.gmu.edu
Phone : 703-993-1651
Fax: 703-993-1638}

Information and Software Engineering Department
George Mason University
4400 University Drive
Fairfax, VA 22030

Abstract:

The architectural model of a system provides a high level description of a system in terms of components and connectors that coordinate the components to meet global requirements. Given a set of components and a scenario-based representation of the required application specific interaction requirements between the components, the application architecture design introduces connectors that provide the application level glue to coordinate the given components. Though the connectors can be introduced to serve multiple purposes, this paper focuses on the role of the connectors to provide causal connection between the components and coordinate the component behaviors to satisfy the scenario specific ordering constraints. For a given composition, the analysis problem is ensuring that i) the causal connection is consistent with the component interfaces and the scenario requirements, and ii) the ordering constraints specified by the scenario is satisfied by the composition under the causal mapping introduced by the connector. This paper presents an automated method to address the above problems. The method takes as input 1) the required interaction scenarios captured as sequence-diagrams that model the event-flows between the components, and 2) an application architectural model of the system in terms of components and connectors whose behaviors are modeled by finite-sate machine models. The method first checks for consistency with respect to the required causal connection and checks via symbolic simulation whether the required ordering constraint is met. The method has been implemented in an extended UML-based design and analysis environment to support component-based software architecture modeling and analysis.

Keywords: Scenarios, components, application Architecture, approximate model checking, UML,

Scenario-Driven Analysis of Component-Based Software Architecture Models

1 Introduction

Recent developments in component-based standards and technologies (COM, DCOM, and CORBA) have led to the increase in the development of component-based software systems [Tho98]. A major problem in composing component-based system is providing assurance that a given composition of components does in fact meet a given set of global integrity constraints (such as coordination constraints and structural invariants) and quality constraints (for example, evolvability, reliability, and security). Recent research on architectural models [MT97] of such systems have led to defining high level models needed to reason about such properties.

The architectural model of a system provides a high level model of the system in terms of components that do the computation and connectors that causally connect the components [Luc95] and coordinate their interactions to satisfy global constraints [Fr94, AG94]. The architecture research community has made considerable progress on the development of architecture description languages (ADLs) [Luc95, Sha95, MK96, AG97, GMW97] that capture the structure of the system in terms of the components that interact via connectors and capture key component and system level properties. The architectural model of the system provides a high level of abstraction to the different stakeholders of the system to reason about its properties (functional and non-functional) and for making extension, customization, and evolution decisions. Recent research has focused on specific ADLs to capture different properties in the architecture [BG98] and their analysis.

In composing components to meet specific requirements, we can consider architectural models from at least two perspectives - the application perspective and the solution perspective. In the application perspective, the model makes explicit the application specific usages, component interfaces and application specific global properties and constraints. The solution perspective captures the mechanisms that realize the application level components and connectors typically expressed in some style (e.g. pipe-&-filter, explicit invocation, implicit-invocation, etc.).

In this paper we focus on the application specific perspective where the components and their ports (interfaces) are abstracted with respect to a given set of application specific concerns. In composing a set of components for a specific set of needs, we view the application level architecture model as defining connectors (synonymously called coordinator in this paper) that causally link (and hence compose) the component interfaces and coordinate their interactions to meet the global integrity constraints. The problem is assuring that the connector does provide the necessary causal connectivity and that the composition satisfies a given set of global application constraints.

This paper presents a scenario-driven approach to addressing the above problem. Scenarios provide a natural way to express required behavioral invariants in an example-oriented manner. They can be used to express non-functional constraints [KABC96]. The scenarios that we consider are scenarios of interactions between the components being composed. The scenario is represented as a sequence diagram capturing the interaction relationships between components in terms of event-flows and the relative ordering of such flows. Figure 1. shows a simplified example of a scenario of interaction, expressed as a sequence diagram, between a buyer and a seller component that specifies the usage of the composition to support goods ordering and payment. The sequence diagram specifies the flow of events[1] between the components. Given such a requirement the application level compositional architecture design defines coordinators that compose the component interfaces to meet the interaction requirements.

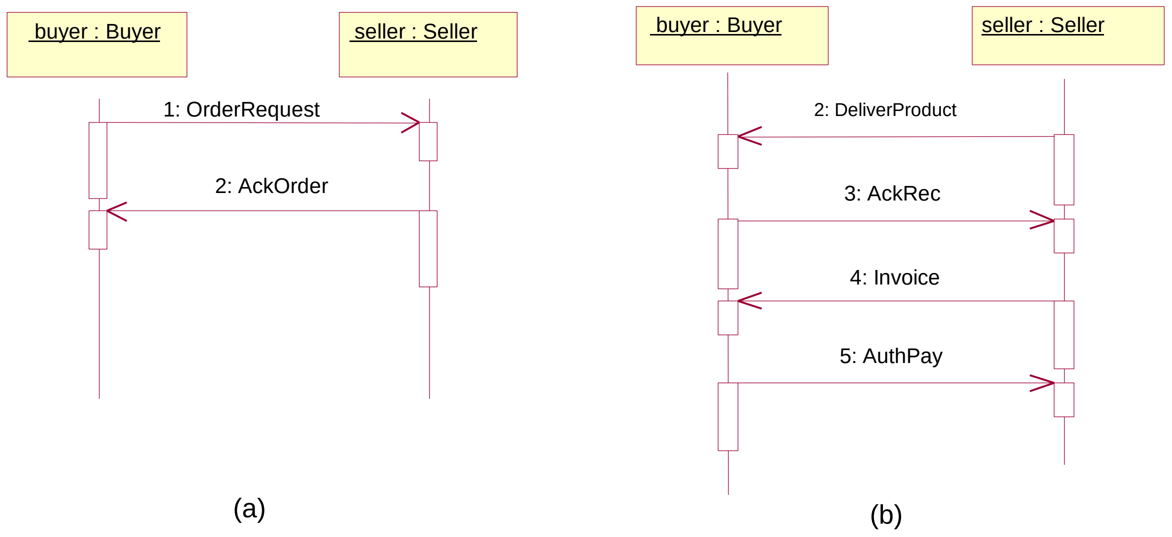

Figure 1: A sequence diagram based representation of the required interaction between the buyer and the seller component.

1.1 Problem and Key Ideas

The sequence diagram specifies two major behavioral constraints that the coordinator-based composition of the components must meet: a) *Causal connection constraint.* Messages or events originatng from the source component must be causally connected to events at the destination component. For example in Figure 1(a) the OrderRequest event from the Buyer must be mapped to some event on the seller component interface, here the RecieveRequest event. Here we make the assumption that the coordinator does not introduce new events. The causal constraint is met if the coordinator defines a mapping of the event from the source to an event that belongs to the destination interface. b) *Ordering Constraint.* Under the mapping induced by the coordinator, the sequential ordering of events relative to each component must be ensured under composition.

The paper presents an approximate approach to analyzing the constructed models against specified scenarios. The key steps in analyzing for the above constraints are:

- *Translation of a specific global behavioral interaction scenario as a set of localized event traces relative to each component.* From Figure 1(a), the subtrace related to buyer is given by the sequence ([<To=Seller, Event =OrderRequest> ; <From = Seller, Event =AckOrder>][2]. Note that multiple

[1] The interaction between the components in a sequence diagram are solely interpreted as event-flows. The directionality of the flow from the source to the destination captures the event produced by the source. The vertical timeline captures the ordering of the flows.

[2] The notation $a_1;a_2$ is used to denote that a_2 follows a_1.

components may be engaged in an interaction. The set of event traces local to each component then defines the history of interactions with the rest of the system via the coordinator interface.

- *Coordinator as an event mapper.* To satisfy the causal connection constraint the coordinator must map producer component events to consumer component events. The individual traces for a component are checked for existence of such mappings. It is to be noted that for a static connector, the event is mapped uniquely to a destination component event across all scenarios. For a dynamic connector (as specified here using FSM models), the source event may get mapped to different events depending on the state of the connector.
- *Testing for each trace by symbolic execution of the model.* The ordering constraint is satisfied if the ordering specified by the localized traces can be satisfied. The method checks the consistency of each local trace by symbolic simulation of the composition in an incremental manner.

The following sections describe the representations used and briefly describe the method based on the above ideas.

2 Representations

2.1 Architectural Modeling in UML

We model architectures of component-based systems using the extensionibility feature of the Object-oriented Modeling Standard UML [UML97]. We exploit the stereotype concept in UML to define the architectural description language (ADL) specific constructs. Figure 2 shows the architecture meta-model elements and relationships represented using UML stereotypes. The architecture meta-model builds on the architectural modeling constructs developed in the ACME architecture interchange language [GMW97]. We introduce extensions to the ACME elements that are necessary to model component and connector behavior.

As shown in the meta-model, the behavior object is modeled as a first-class object that is attached to a component or connector via an association relation. The behavior can be modeled as a finite state machine or as a process (CSP [Hoa85]) or as constraints over events (using propositional temporal logic). In this paper we consider only finite state machine models of the component and connection behavior. The ports of component and roles of components have a protocol signature[3] specified by the in/out stereotyped events suffered/generated by the port or role. The semantic relationship between a components port protocol and the component's behavior is that the behavior of the component defines the computation (similar in spirit to that of WRIGHT [AG97]) by specifying how input events are mapped to other events or actions (observable or internal). The component behavior may also be used specify coordination of port behaviors locally. Similar semantic relationship holds between a connector's behavior specification and the role specifications – here the connector behavior specifies the coordination of the roles. The major difference is that the connector does not generate any application specific events (i.e. all events are determined by the external context consisting of the components being coordinated).

[3] ADL's like Wright [AG97] consider CSP based specifications of port and role behaviors as well as the component and connector behaviors.

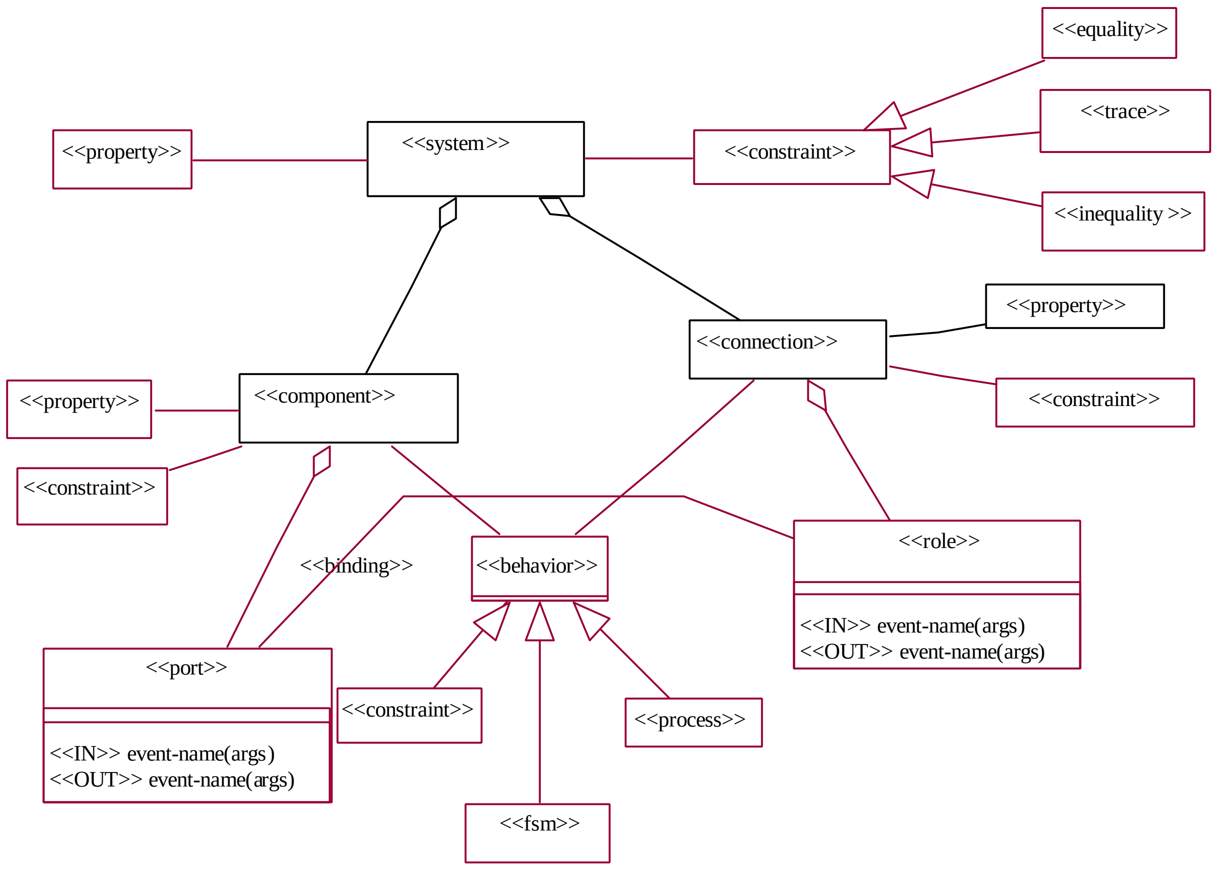

Figure 2: Architecture meta-model defined using stereotypes in the Unified Modeling Language (UML). The modeling constructs are based on extensions to constructs in ACME [GMW97].

2.2 Composition Usage Specification: Component Interaction Diagrams

Components get composed to serve specific global functions that require their behavioral interactions. The global functions of a component-based system can be behaviorally specified using a set of scenarios that define the desired behavioral interaction relationships between the components. In UML such scenarios are best captured using object-interaction diagrams called sequence diagrams. The sequence-diagram specifies the objects (here component instances) that are involved in the interaction, event-flows between the components and a temporal ordering on the flows. Figure 1 shows an example of the sequence diagram for the buyer and seller component composition example. It is to be noted that multiple sequence diagrams may be required to completely specify a function. The event-flow link from the source to the destination is labeled with the event from the source.

2.3 Coordinator based Application Architecture

Coordinators have typically been used to coordinate solution level components. In a similar spirit we consider the application architecture as consisting of components and coordinators that are modeled strictly from an application perspective [NACO97]. The coordinators specify the application specific glue that composes the component behaviors to achieve global integrity constraints. The components and coordinators have ports and roles respectively and the architecture specifies the binding between the ports and the roles. It is worth noting that in mapping the application architecture to solution architecture, the coordinators can be realized by the connectors at the solution level or they can also become first-class

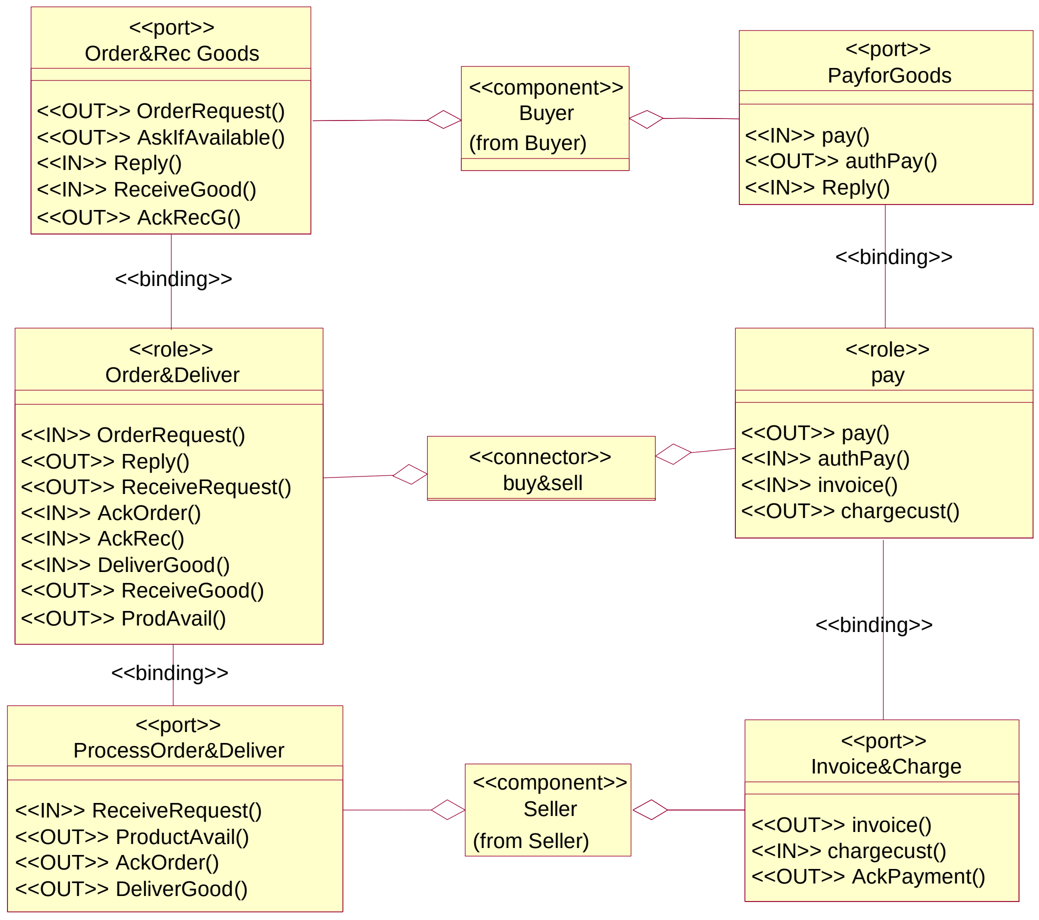

Figure 3: The Application architecture for buyer and seller component composition example.

components at the solution level. Figure 3 shows the composition of the buyer component and the seller component based on the buy&sell connector.

The components and connectors are specified using finite-state machine models. Figure 4 shows the state-chart diagram representation of the buy&sell connector behavior. The transitions between the states specify the causal connection between a given input event from a source role to an output event in a destination role. For simplicity, the machine model in the figure only shows distinctions between buyer (b) or seller (s) role.

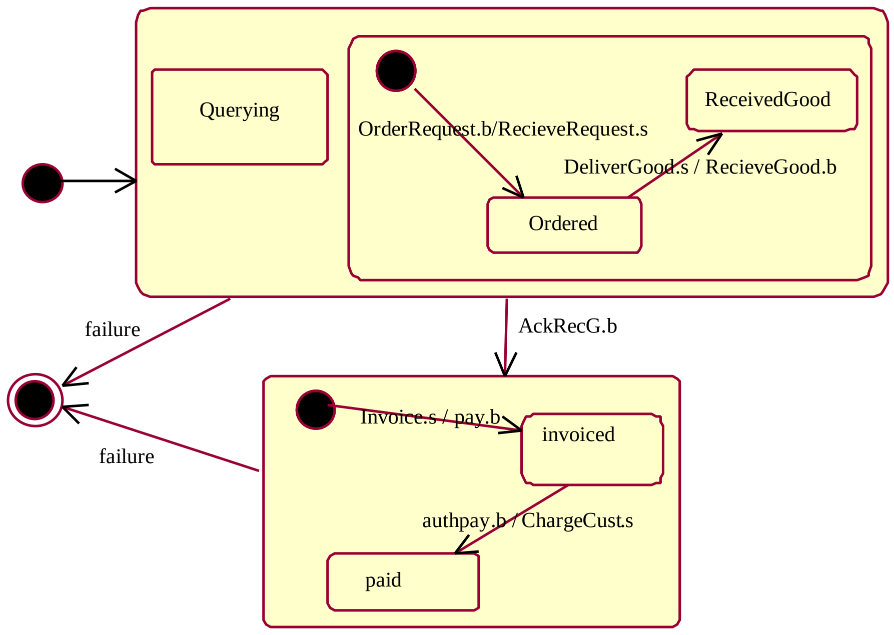

Figure 4: A Finite-State machine model, using state-charts, of the buy-&-sell coordinator behavior

Given the usage specification of the composite system as interaction scenarios that involve the coordinator, the problem of analysis of the design is to check if the coordinator-based design respects the behavior interaction relationships required by the scenarios by satisfying the causal connectivity constraint and the ordering constraint.

3 Scenario Driven Coordination Model Checking: The Method

The method for checking the composed model translates the interaction scenario specification into local traces, checks for consistent mapping, and then does symbolic simulation of the coordinator based composed model to determine if sub-sequences of the required behavioral interaction are accepted by the model. The method is based on the simple premise that consistency of the translated local specifications with respect to the coordinator-based refinement ensures satisfaction of the global interaction requirement. An informal argument for such a premise is as follows: a) the sequence diagram behavior specification holds iff the local behavior specifications hold. b) for the composed model to be consistent with the required behavior, the model should entail the behavior and hence in turn entail the individual local specifications. Hence the method systematically checks for satisfaction of the causal connection constraint and the ordering constraint relative to each local trace specification. We briefly describe the three basic substeps of the checking process.

- *Translation of global to local.* This step involves taking the sequence diagram specification and generating the local traces based on the timeline for each component. Each element in the generated local sequence is represented by the tuple [<From/To> <Event>]. For example, the local trace for the seller component obtained from the specification in Figure 1(a) is ([<From=Buyer, Event =OrderRequest>; < To = Buyer, Event =AckOrder>]. An observation to be made here is that, when only two components are involved in an interaction, the local-traces are symmetric and hence checking for one local trace entails satisfaction of the other.

- *Checking for causal connection.* The causal connection checking involves checking for existence of coordinator mappings that connects a source component event with a destination component event in a

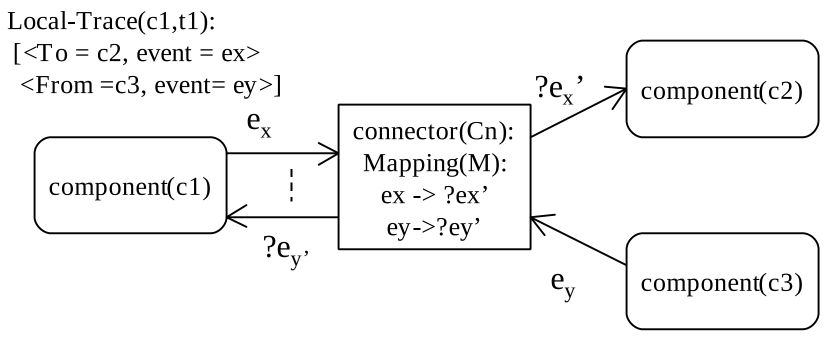

Figure 5: The causal connection problem with respect to local trace t_1 for component c_1.

local specification. Figure 5 shows the causal connection problem schematically for a given local trace for component c_1. Simple checking of consistency of the port and role protocol signatures for the given architectural binding is not sufficient since such checking does not provide the mapping information necessary to do the consistency checking. The checking involves obtaining the mappings for the given source events from the finite-state machine specifications of the coordinator, here e_x and e_y to $?e_x'$ and $?e_y'$ respectively, and determining consistency of e_x' and e_y' with respect to the component port specifications. The mapping is derived from the understanding that the transitions in a coordinator specify the causal link between the input event in a role (hence the output event of a port bound to the role) and the output event in a role (and hence the input event of the port bound to the role). For example, the connector specification in Figure 4 causally maps the OrderRequest event from the buyer to the RecieveRequest event of the seller (skipping the ports and roles for brevity). For a dynamic connector specified by an FSM, each mapping specifies the parent state, the transition from which generates the output event.

- *Ordering constraint checking.* The previous step, if successful, yields one or more possible mapping relevant to the local specification. The ordering constraint checking involves determining existence of a path of events that starts with the starting event in a trace and leads to the subsequent events in the localized specification, where intermediate events maybe generated by other components via the coordinator. Since for a given input event to the coordinator, the output event is dependent on the coordinator state and multiple state transitions may exist for that event, for completeness the method must explore all such paths. We take an incremental approach. The method does approximate checking for a given trace in the sense it checks of existence of a path that either succeeds or fails via divergence. We consider the model as failing under divergence when the model accepts an event to or from the component that is not consistent with the trace specification. For example, if the simulation trace for the buyer yields OrderRequest followed by Invoice event from seller, then the model is divergent relative to the ordering trace of OrderRequest followed by AckOrder event from seller. The basic steps of the method are shown in Table 1.

Table 1: The Event Ordering Checking Method.

```
Method(Tr, C, R, M)
  Tr = trace = sequence of  input/output events {$e_1$; $e_2$ ; ..}
  C = component relative to which Tr is being checked
  R = rest of architectural model consisting of other components and coordinators;
  M = set of Mappings applicable to events in Tr
  Let,
     S   = Simulation State = nil /* current active states indexed by component and connectors */
Steps:
1. Do until empty(Tr)
Begin:
1.1 Xe = get-next-event (Tr) /* The get is destructive */
1.2 Case:
    [a]  Xe.FROM = Component($C_i$) and $C_i \in$ R
        Begin:
          Get-input-to-comp(Xe, C, ?Xe', M, S)
            /* Using the applicable mappings M, map the Xe to ?Xe' consistent with S; update state S for connector */
          Run_FSM(C, S)  to  find next events (Eo) for input(?Xe')
            /* Symbolically execute the FSM for C consistent with S and Update S for C*/
          Let $Xe_n$ =  read-next-event(Tr) /*  non-destructive read */
          If   $Xe_n$.TO = component(Cy) & Cy $\in$ R
               And  fail-to-match-any ($Xe_n$, Eo)
          Then  FAIL(Xe);
          Else   CONTINUE.
        End.
   [b] Xe.TO = component($C_i$) & $C_i \in$ R
        Begin:
         /* The The repeat loop searches event traces(P) starting with event Xe from C until  it finds a
           path(P)  such that the path has an  event Xy where Xy originates from some Ci and
           there exists an applicable Mapping for Xy to C */
           Repeat [Obtain-next-sim-path(Xe, S, M, P) ]
           Until  [event(Xy) in P & Applicable-Mapping-To(Xy, C) ]
           Let $Xe_n$ = read-next-event(Tr);
           If   matches($Xe_n$, Xy)
            Then Continue;
          Else  FAIL(Xe).
         End;
 End-Do;
```

The does incremental checking by simulating for one event at a step and checking for consistency of the generated events with respect to next event in the sequence. The cost of checking for all event-traces may be very high for complex compositions and large number of scenario specifications. One option that can be explored to reduce the cost is by define using the traces to generate a lattice under trace sub-sumption relation and start checking systematically with smallest sequence. Any failure with a specific subsequence can be used to infer failure of super-traces.

3.1 Object-Oriented Architecture Modeling and Analysis: Support Environment

We have done an initial prototype of scenario-driven analysis component as part of a suite of tools for collaborative component-based software architecture modeling and analysis environment [Bo98]. The environment is built via extending Rational Rose tool [Ro98] with tools to support for a) Capture of component-based software architecture models following the meta-model defined in Figure 2. b) Capture of dependencies between architectural decisions and architectural requirements via the WinWin meta-model [Bo98, BBHL95]. c) What-if scenario driven analysis for change-analysis using model of the dependencies captured by the WinWin artifacts. d) Behavior interaction scenario driven analysis of composition based on the method presented in this paper. The current application of the environment and tools involves modeling and analysis of component-based simulations [JB98] in the JSIMS domain.

4 Related Work

The work presented in this paper is related to the current work on integrating architectural modeling languages with Standard design method and architecture analysis. Our work on use of UML to define the meta-model for capturing architectural models is similar in spirit to that of Robbins et. al [RMRR98]. The work presented here exploits the stereotype extensibility feature to capture the ACME modeling constructs. Research on analysis of architectural models has been conducted based on the underlying semantic model in the ADLs. The work on WRIGHT [AG97] is based on CSP [Hoa85] models of components and connectors and uses FDR [For92] model-checking tool for deadlock analysis. The work on Rapide [Luc95] is based on partially ordered event model (POSET). Rapide is an executable ADL and hence analysis for event-orderings is based on the POSET graphs resulting from the simulation of the architecture model. There has also been recent progress on compositional approaches to analysis of architectures [CK96] for safety and liveness properties. Other formal methods based approaches to model checking of software systems [WF95, JM96] do not consider model of the system at the architecture level in terms of components and connectors. The work presented in this paper builds on the symbolic model checking concepts developed in SMV [McM93] and used in [WF95] to apply to analyzing component compositions for required scenarios of interactions formalized as sequence diagrams.

5 Summary and Future Work

This paper presents a systematic method for scenario-driven analysis of component-based software architectural models. We consider the architectural model from the application perspective. More specifically the paper considers coordinator based composition of independent components with the objective to satisfy the interaction requirements specified by given scenarios expressed as sequence diagrams. For a given composition, the paper defines the analysis problem as ensuring that i) the causal connection is consistent with the component interfaces and the scenario requirements, and ii) the ordering constraints specified by the scenario is satisfied by the composition under the causal mapping introduced by the connector. This paper presents an automated method to address the above problems. The method takes as input 1) the required interaction scenarios captured as sequence-diagrams that model the event-flows between the components, and 2) an application architectural model of the system in terms of components and connectors whose behaviors are modeled by finite-sate machine models. The method first checks for consistency with respect to the required causal connection and checks via symbolic simulation whether the required ordering constraint is met. The method has been implemented in an extended UML-based design and analysis environment to support component-based software architecture modeling and analysis. Current and future work is aimed at a) improving the efficiency of the approach by considering the lattice of traces, b) applying the tool to consider solution architectures as well, c) improving the tool via experimenting in the JSIMS domain, and d) Usage of the tool to automatically detect issues based on Winconditions that identify the desired scenarios.

6 Acknowledgements.

The author thanks Xiaoqing Zhou and Ping Chen for their contribution to the development of the prototype implementation in the Rationale Rose environment. The author also thanks Jesse Aaronson for the modeling work in the JSIMS domain where the method is currently being applied and debugged.

7 References

[AAG93] G. Abowd, R. Allen, D. Garlan. "Using Style to Understand Descriptions of Software Architecture", Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering, Software Engineering Notes, ACM Press, December 1993.

[AG94] R. Allen and D. Garlan, "Formal Connectors", CMU-CS-94-115, Carnegie Mellon University, March 1994.

[AG97] R. Allen and D. Garlan, "A Formal Basis for architectural Connection", ACM Transactions on Software Engineering and Methodology, 1997.

[BBHL95] Boehm, P. Bose, Ellis Horowitz and Ming June Lee "Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach", IEEE Proceedings of the 17th ICSE Conference, 1995.

[Boe98] P. Bose, " Change Analysis in an Architectural Model: A Design Rationale Based Approach", Submitted to ISAW3, 1998.

[BG98] B. Spitznagel and D. Garlan, "Architecture-Based Performance Analysis", 10th Internatnational Conference on Software Engineering and Knowledge Engineering (SEKE98), submitted.

[CK96] S. C. Cheung and J. Kramer, "Checking subsystem Safety properties in Compositional Reachability Analysis", ICSE 18, Germany, 1996.

[Fr94] Svend Frϕlund, "Coordinating Distributed Objects: An Actor-Based Approach to Synchronization", MIT Press, 1996.

[Gar98] D. Garlan, " Higher Order Connectors", Workshop on Compositional Software Architectures, Monterey California, 1998.

[GAO95] D. Garlan, R. Allen, and J. Ockerbloom, "Architecture Mismatch: Why Reuse is so hard", IEEE Software, pp 17-26, November 1995.

[GMW97] D. Garlan, R. Monroe, and D. Wile, "ACME: An Architectural Description Interchange Language", Proceedings of CASCON 97, November 1997.

[GS96] D. Garlan and M. Shaw. "Software Architectures: Perspectives on an Emerging Discipline", Addison Wesley Publishers, 1996.

[Hoa85] C. A. R. Hoare, "Communicating Sequential Process", Prentice Hall 1985.

[JB98] J. Aaronson and P. Bose, " Model Based Simulation Composition", submitted to Automated Software Engineering Conference, Hawaii, 1998.

[HM96] Constance Heitmeyer and D. Mandrioli, "Formal Methods for Real-Time Computing", John Wiley, 1996.

[KABC96] R. Kazman, G. Abowd, L. Bass and P. Clements, "Scenario-based Analysis of Software Architectures", IEEE Software, November 1996, pp 47-55.

[Luc95] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, and D. Bryan, "Specification and Analysis of System Architecture Using Rapide", IEEE Trans. Software Engineering, vol. 21, no. 4, pp. 336-355, April 1995.

[McM93] K. L. McMillan, "Symbolic Model Checking", Kluwer Academic Publishers, 1993.

[MK96] J. Magee and J. Kramer, "Dynamic Structure in Software Architectures", In proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering.

[MQ94] M. Moriconi and X. Qian. "Correctness and Composition of Software Architectures", Proceedings of the Second ACM SIGSOFT Symposium on Foundations of Software Engineering, Software Engineering Notes, ACM Press, December 1994

[MT97] N. Medvidovic and R. Taylor, "A Framework for Classifying and Comparing Architecture Description Languages, ESEC/FSE 97.
[NACO97] G. Naumovich, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil, "Applying Static Analysis to Software Architectures", ESEC/FSE97.
[PW92] D. Perry and A. Wolf, "Foundations for the Study of Software Architecture", ACM SIGSOFT Software Engineering Notes, Vol. 17, 4, October 1992.
[RMRR98] J. Robbins, N. Medvidovic, D. Redmiles, and D. Rosenbloom, "Integrating Architecture Description Languages with a Standard Design Method", Second EDCS Cross Cluster Meeting, Austin, Texas, 1998.
[Ros98] A. W. Roscoe, "The Theory and Practice of Concurrency", Prentice Hall, 1998.
[Ro98] Rational Rose 98, Tool and Documentation, Rational Software Corporation.
[Sha95] M. Shaw, R. DeLine, D. Klein, T. Ross, D. Young, and G. Zelesnik, "Abstractions for Software Architecture and Tools to Support Them", IEEE Transactions on Software Engineering, April, 1995, 314-335.
[Tho98] C. Thompson, editor, "Workshop on Compositional Software Architectures", Monterey California, January 6-8, 1998.
[UML97] UML Semantics, version 1.1, Documentation, Rational Rose Corporation.
[WF95] J. Wing and M Vaziri-Farahani, "Model Checking Software Systems: A Case Study", Software Engineering Notes, 20(4):128-139, Oct 1995. Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering.