An Approach to Web Services Oriented Modeling and Validation

Yujian Fu, Zhijiang Dong, Xudong He
Florida International University
Miami, FL, 33199
{yfu002,zdong01,hex}@cis.fiu.edu

ABSTRACT
Web services provide a language-neutral, loosely-coupled, and platform independent way for linking applications within organizations or enterprises across the Internet. Web services communicate with each other via XML format messages. This paper presents a web service architecture model, Service-Oriented Software Architecture Model (SO-SAM), which is an extension of SAM (Software Architecture Model [16]) to the web service applications, as well as a validation of the model and a case study. SO-SAM is an executable architectural model incorporating Predicate Transition Nets with the style and understandability of component-based concepts. SO-SAM describes each web service in terms of component and service composition in terms of connector separately. We believe that SO-SAM facilitates the verification and monitoring of web services integration since SO-SAM fits the distributed nature of modern composite web services. In order to validate the model against system properties, we rewrite the SO-SAM into the XML format, and validate SO-SAM using a SAM support tool, SAM parser. Finally, a case study of the validation of the model is demonstrated.

Categories and Subject Descriptors
D.2 [Software]: Software Engineering; D.2.4 [Software Engineering]: Software/Program Verification—formal methods, validation; D.2.11 [Software Engineering]: Software Architecture—Languages

General Terms
Service Oriented System Design, Verification

Keywords
Service oriented system specification, verification and validation, software architecture model, Petri nets, Temporal logic

1. INTRODUCTION
The emerging web service paradigms opens a new way of web application design and development through integrating independently published web services components to conduct new business transactions. Accordingly, a web service-oriented system refers to a system that integrates multiple web services automatically and transparently. While several established and emerging standards bodies (e.g., [8, 6, 5, 1, 2] etc.) are rapidly laying out the foundations that the industry will be built upon, there are many research challenges behind web services that are less well-defined and understood [6] for the large number of web service application design and development. Software architecture is critical to decide the success of a web service-oriented system. In providing the high level perspective, triggering the right refinement to the implementation, controlling the quality of services of products and offering large and general system properties software architecture plays an essential role. Currently most research focuses on the integration and automatical composition of web services but pays less attention to the whole structure and behaviors of web services on composition and verification. Recently, a number of Web Service-oriented Architecture Description Language (WSADL), such as Web Services Description Languages (WSDL) [8], Web Services Flow Language (WSFL) [17], Business Process Execution Language for Web Services (BPEL4WS) [1], Web Service Choreography Interface (WSCI) [6], XLANG [19], etc. have been developed and applied in both the academic and industry. However, these WSADLs either merely focus on static descriptions of web service functional activities as a whole, or concentrate only on the behavioral integration between web services. Furthermore, less of these current WSADLs support dynamic verification and monitoring of the integrated system.

SAM [16] is an architectural description model based on Petri nets [20] that is well-suited for modeling distributed systems and temporal logic [18] for specifying system properties. In this paper we propose a service-oriented architecture model, SO-SAM, which extends SAM to the web service applications and overcomes the drawbacks of current web service description languages. SO-SAM is an executable specification language based on the Predicate Transition Net (PrTN) [20] and the temporal logic [18] specification of system properties. Supporting modern software engineering philosophies equipped with component-based notations, SO-SAM provides an approach to specify and monitor the dynamic composition of web services and integration of service-oriented systems.

The remainder of this paper is organized as follows. In Section 2, related work is discussed. In section 3, we review SAM with predicate transition nets and temporal logic for high-level design. In section 4, we present SO-SAM specification. After those, a verification approach is presented in section 6 and case study is demonstrated in section 5. Finally, we draw conclusions and describe future work in section 7.
2. RELATED WORK

Many web service description languages provide an architectural perspective of web applications. Generally, all of the proliferating work is built upon eXtensible Markup Language (XML) [3] technology. We simply describe some languages in the following.

- SOAP (Simple Object Access Protocol) [5] is “an XML based protocol for information exchange designed to allow communication in a decentralized, distributed environment...” [5]. There are three general parts: (1) an envelope that defines a framework for describing what is in a message and how to process it, (2) a set of encoding rules for expressing instances of application-defined data types, and (3) a convention for representing remote procedure calls and responses. SOAP can potentially be built on top of any transport layer, e.g., an HTTP-based infrastructure [5].

- BPEL4WS [1] (Business Process Execution Language for Web Services) [1] is an XML language that supports process-oriented service composition and used to specify business processes and business interaction protocols. It is originated from XLANG and WSFL as a standard for web services flow specification. The model and XML-based syntax defines the interactions between process and its partners that occur using Web service interfaces. It also defines the states and logic of coordination between these interactions as well as provides a way of handling exceptions [22].

- WSDL (Web Services Description Language) [8] is “an XML format for describing network services based on a standard messaging layer like SOAP. A WSDL description defines services as collections of operations or ports, accessible over network...” (21]). In WSDL, the abstract definitions of operations and messages are separated from their concrete realization and data format. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data, and port types (abstract collections of operations). However, WSDL can only specify limited information of a web service, such as the function names and limited input and output information [13].

- WSFL (Web Services Flow Language) [17] was proposed to describe compositions and interactions of services as well as usage pattern of services to achieve the business goal. WSFL specifications tell us how different services ought to be invoked, e.g., in terms of ordering and parallelism among them, but they do not tell us whether a particular service that is bound in the workflow will in fact deliver the right interactions.

- DAML-S [9] (DARPA Agent Markup Language) extends XML and the Resource Description Framework (RDF) to provide a set of constructs for creating machine-readable ontologies and markup information. DAML-S is a services ontology that enables automatic service discovery, invocation, composition, interoperation, and execution monitoring.

- WSCIL (Web Services Conversation Language) [7] has recently been developed as a complement to WSDL. Whereas the latter specifies how to send messages to a service, it does not state the order in which such messages are allowed to be sent. This issue is addressed in WSCIL, which defines legal sequences of document exchange between web services.

- WSCI (Web Service Choreography Interface) [6] is a XML-based language for description of message exchange carried by web services using collaborative business process or workflows. It defines the flow of messages exchanged by a stateful web service. It is used to describe a service in a way that other web services can interact with it over the message exchange by specifying the temporal and logical dependencies. Although it provides a message-oriented view of the process, it does not define the internal behavior of the web service or the process.

However, these WSADLs focus on the topological description and concentrate on describing interactions between web services components. They lack the capability to describe the hierarchical functionality of the components. There is little concern about expressing dynamic behaviors of the defined system. Furthermore, none of these current service description languages support dynamic verification and monitoring of the system integrated.

3. PRELIMINARIES

SAM (Software Architecture Model) [16], an architectural description language, not only provides means to define structure and behavior of software architecture, but also provide means to specify behavioral properties for components and connectors that should hold in the architecture by Petri nets.

SAM [16] has dual formalisms underlying – Petri nets and Temporal logic. Petri nets are used to describe behavioral models of components and connectors while temporal logic is used to specify system properties of components and connectors.

SAM architecture model is hierarchically defined as follows. A set of compositions \( C = \{C_1, C_2, ..., C_i\}\) represents different design levels or subsystems. A set of component \( C_{n_1}\) and connectors \( C_{n_2}\) are specified within each composition \( C_i\) as well as a set of composition constraints \( C_{n, k}\), e.g. \( C_i = \{C_{n_1}, C_{n_2}, C_{n_3}\}\). In addition, each component or connector is composed of two elements, a behavioral model and a property specification, e.g. \( C_{i, j} = (S_{i, j}, B_{i, j})\). Each behavioral model is described by a Petri net, while a property specification by a temporal logical formula. The atomic proposition used in the first order temporal logic formula is the ports of each component or connector. Thus each behavioral model can be connected with its property specification. A component \( C_{n_1}\) or a connector \( C_{n_2}\) can be refined to a low level composition \( C_i\) by a mapping relation \( h\), e.g. \( h(C_{n_1}) \) or \( h(C_{n_2})\) = \( C_i\).

SAM gives the flexibility to choose any variant of Petri nets and temporal logics to specify behavior and constraints according to system characteristics. In our case, Predicate Transition (PrT) net [14] and linear temporal logic (LTL) are chosen.

In summary, although our work was strongly influenced by SAM, we have enhanced the state of the art by supporting modern software engineering philosophies equipped with component-based and object-oriented notations and applied to web services-oriented systems, as well as integrated with WSDL and XML.

4. SO-SAM SPECIFICATION

In order to specify both the static and dynamic architectural features of a web services-oriented system, we present a Service Oriented Software Architecture Model (SO-SAM) as an executable architectural specification language. Two web service components are composed through connectors. SO-SAM specifies a web service-oriented software architecture as a set of connected architectural components described by PrT nets and service properties described by temporal logic formulae. The architectural components correspond to functional units in the system. With the hierarchical nature of SAM model one architectural component may in turn be
composed of several smaller architectural components. The entire system can be viewed as a highest-level architectural composition, which is also a component. Each architectural component is either statically or dynamically realized by a web services component. Architectural components are connected to each other via XML message passing through Simple Object Access Protocol (SOAP) [12]. The behavior of the connection is specified by SAM architectural connectors. The message passing mediates the interactions between architectural components via the rules that regulate the component interactions. In our model, connectors carry the tasks of service compositions. Thus our model supports both executable and dynamic web service compositions.

**DEFINITION 1 (SO-SAM).** Service-Oriented Software Architecture Model, SO-SAM, is defined by a set of components \( C_m \), a set of connectors \( C_c \), and a set of constraints \( C_s \), e.g. \( \text{SO-SAM} \triangleq < C_m, C_c, C_s > \), with following regulations:

- Two components can communicate with each other only through connectors, which model service integration.
- The message in each port is defined in the XML format and passed through SOAP protocol.
- The component follows the definition of WSDL or XML.
- The dynamic behavior of a component is defined by a service net, which is focusing on its internal operational nature.

In section 3, we know that the ports in each component are either input ports or output ports. In the extension to web applications, ports are used to transfer messages among services, same as in SAM model. However, we regulate messages as a tuple with the information of service name, service description, location, URL, etc., so that the message carries service information.

A component is composed of the above ports that carry service information, behavior description and property specification. The behavior of a component is defined by a Petri net, which is called a service net. In the service net, tokens in a place has to have specific sort to be consistent with the above port and message definition. A basic component is one that does not have sub-components and non-empty connectors, otherwise, it is a composition. A composition is a composite service. The relation between a composition and its subcomponents and connectors is defined by a mapping function \( f \). Mapping function \( f \) is a set of maplets from super component/connector’s identities to subcomponents’ (connectors’).

Service integration can be done through connectors. Connectors have the same definition as in SAM. The Petri net for a connector is a regular petri net that describe the integration and composition of services. A connector cannot be a composition.

**DEFINITION 2 (PORT).** Ports in SO-SAM are communication interfaces of each services and graphically represented by semicircles. Messages in ports are modeled by tokens. The sort of each token is defined by two parts; service sort \( S_S \) and data sort \( S_D \). Service sort is a four tuple, \( < \text{SN a m e}, \text{SD e s c}, \text{Loc}, \text{URL} > \), data sort is the regular sort defined in SAM [15].

Thus we have sort \( S \) of a port is defined by \( S \triangleq S_S, S_D \).

Service sort \( S_S \) is a service query indicating a service name, location, description, URL, etc.. Service sort must be carried by all tokens in the service net.

**DEFINITION 3 (COMPONENT).** Each component \( C_m \) in SO-SAM is defined by a tuple, component name \( C_m \) ID, mapping function \( f \), set of ports \( P_t \), that is composed of the set of input ports \( P_{ni} \) and the set of output ports \( P_{fo} \), the set of initial ports \( P_{ni} \subseteq P_t \), the set of finial ports \( P_{fo} \subseteq P_t \), a service net \( S N \), and a set of temporal logic formulae \( ST \), e.g.:

\[
\begin{align*}
C_m & \triangleq < C_m \text{ID}, f, P_{ni}, P_{fo}, P_t, S N, ST >, \\
\text{where} \ f & \text{ is a mapping function : } C_m \text{ID} \rightarrow \text{subC_mID}, \text{ and } \end{align*}
\]

where \( f \) is a mapping function \( f \), where each sort must contain \( S_S \). Mapping function \( M_p \) associates each port to a place in the service net.

**DEFINITION 4 (SERVICE NET).** A service net is a Petri net defined by 8-tuple, \( \text{SN} \triangleq < P, T, F, \phi, R, L, M_p, M_p >, \) where

- \( P, T \) are finite set of places and transitions; \( F \) is flow relation: \( P \times T \cup T \times P \).
- \( \phi \) is sort assignment: \( P \rightarrow o(S_i) \) ([15]), but sorts \( S \) are extended to carry service information.
- \( R \) (guard function), \( L \) (label function), and \( M_p \) (initial markings) follow the definitions in the paper [15].
- \( M_p \) is place mapping function, \( M_p : P_t \rightarrow P \)

where each sort must contain \( S_S \). Mapping function \( M_p \) associates each port to a place in the service net.

**DEFINITION 5 (WEB SERVICE).** A web service is defined by a basic component or a composition (collection of components and connectors).

The behavior of each web service \( S_i \) in SO-SAM is defined by a service net \( S N \), which must starts when the initial ports \( P_{ni} \) has messages and ends when the final ports receive messages. The properties are defined using a set of temporal logic formulae \( ST \).

The composition of different web services can be modeled by connectors.

5. PRESENTATION OF A CASE STUDY ON THE SO-SAM MODEL

This section introduces the case study used throughout the paper. It deals with a simplified online shopping example adapted from [4] and [23]. It’s a basic electronic commerce process of online shopping with credit card transactions. Fig. 1 describes online shopping processing in a free natural language, while an accompanying pictorial scheme provides the details of electronic shopping that precedes the payment processing.

5.1 Properties of correct e-business processes

After studying different kinds of errors discovered during the validation, three property categories have been selected:

1. Structural properties: this kind of properties is closely related to the topology of the model. These properties can be directly verified on the SO-SAM model without animating the transactions. These properties are necessary conditions that ensure the feasibility of the transaction. If one of them is not fulfilled, we can assert firmly that the transaction is erroneous. Among these properties, we cite the following properties:
Customer

Customer browse on the web for some products shown in the website. The customer click on the AddToShoppingCart button (hyperlink), they are taken through a secure shopping cart software and checkout process (hosted on Payment Online secure servers) where credit card transaction information can be entered. After processing the credit card online, authorization responses are then returned to the customer’s to show the transaction status. If the transaction is successful the customer confirm this transaction, the system generates the order and shipping information and inform customer through email or browser. Otherwise, the transaction failed, and the error message is shown to customer through browser.

ShoppingCart

Shopping cart keeps a list of a customer’s items and indexes each item. When the checkout button sends the message to the cart, all items of the customer (identified by customer ID) will be sent to the retailer’s online platform.

Warehouse

The web keeps all available information of the products of retailers, such as product name, category, price, etc. If the quantity of a required item exceeds the storage in the warehouse, the whole transaction is discarded. Otherwise, order can be generated.

OrderProcessing

The web calculates the total price of all items that a customer requested and validates the customer information such as credit card number, credit line, expiration date. It then sends the form of credit card information out and waiting for response. If the credit card information is valid, the order is processed and the related information, i.e., order number, is sent to the customer. Otherwise, the transaction is failed.

CreditCardProcessing

The authorization department checks the credit card information, such as the card has not been reported stolen and there is sufficient credit on the card. Then reports to the retailer’s company.

RetailerBank

The service provided here is collect or refund money from or back to customer.

Figure 1: Natural language description of electronic shopping System

- The services of each outgoing port in a component have same names as the incoming port of its connector.
- Each outgoing port with request message must have been responded either positively or negatively. The positive response determines the service getting the needed response and the transaction can be continue. The negative response indicates an error message, which is an end point. Reaching an error end point means that the service needs to be aborted or guided to some safe program to continue the transaction.
- Each final port must be reachable from the initial port. This property ensures that each service is not superfluous.
- If several final ports exist simultaneously, then these final states are the common states of the transaction. They are one set of final states and cannot be separated. This property ensures that the transaction terminates on only one combined (set of) final state(s), and guarantees that when a (set of) final state(s) is reached, the whole transaction cannot be active.

2. Dynamic properties: the dynamic feature of these properties means that they are related to state changing of the system. The evaluation of the dynamic properties are based on the behavior description – Petri nets. Its verification is achieved on a set of places describing a possible evolution of the system. Among these properties, we cite the following properties:
- Liveness check of web application is necessary to make sure that the service request can get response so that the transaction can continue.
- Deadlock freedom of web application means that operations of a web service do not have circular waiting.

3. Real time properties: these properties aim at evaluating the system in terms of time response. Having constraints on the different activities, we want to be able to determine if the system can answer in a limited time. The constraints that can be stated on activities include delay, period, deadline, etc.

In this paper, we focus on the structural and dynamic properties. We show how we have adapted the SO-SAM for the modeling and the verification of these properties.

5.2 SO-SAM model of online shopping

The top topology of an online shopping system specified in SAM model is given in Fig. 2. Each description block is captured in a component (or a composition). Each composition/component captures a service with the initial and final ports. For instance, service Customer and ShoppingCart is modeled in the component Customer and component Cart, and both components form a composition, a composite service, User in the topology of SO-SAM. In the composition User, a customer starts from request to browse the web (through port request_51), then choose category, select and checkout items. Finally the transaction ends at either getting the order and ship information (port ordNum_15 and shipEmail_5) or payment errors (port checkoutErr_5). In addition, a user can also adjust the order and cart list. All internal operations are represented by the refined inner components, service nets and connectors. Description blocks Warehouse and OrderProcessing are specified in the service component Warehouse and Order, which form another composite service E-Company. An E-company starts at port request_27 to evaluate user requests, and confirm or reject user requests, then submit user information forms and generate orders, etc. Finally, the service component PaymentProcess captures the behavior of the description block CreditCardProcessing. Description block RetailerBank is simply represented in the service component RetailerBank. Because of space limitation, we didn’t show the service net of each component.

6. VALIDATION OF SO-SAM MODEL

In order to validate SO-SAM model, we adapt it to our SAM support tool, which is named SAM parser. The structure of SAM
AspectJ [10] in our case, aspects wrap up functionality code through joinpoints provided by aspect-oriented programming. SAM parser transfer formulæ to the logic engine through a middleware (Logic server).

6.1 Validation support tool – SAM Parser

The input of the SAM parser is a XML file, which specifies SAM structure (such as components, connectors, ports and their relationships) and property specifications. We first translate the SO-SAM architecture to the XML format. The translation rule is based on the structure of the architecture. In the work of [12], we have presented the rules for the SAM translation. SO-SAM model extends SAM model in several aspects, but still keeps the same structure. Thus we can adapt these translation rules to SO-SAM directly.

In the XML file, SO-SAM behavior, the service net, is defined by referring to a Petri Net Markup Language (PNML) [11] file, which is an XML-based interchange format for Petri nets. The logic engine is responsible to construct a piece of pseudo code called monitoring code for each temporal logic formula. A piece of monitoring code is invoked to check if the corresponding formula is satisfied whenever an interesting event occurs. The logic server is a middleware that translates monitoring code to target language, here Java. The SAM parser merges all translated monitoring code for properties of a component or a connector into an aspect. All generated aspects are called runtime verification code.

The SAM parser weaves monitoring code into functionality code by integrating them as aspects. In aspect-oriented programming, AspectJ [10] in our case, aspects wrap up pointcuts, advice, and inter-type declarations in a modular unit of crosscutting implementation where pointcuts pick out certain join points in the program flow, advice brings together a pointcut (to pick out join points) and a body of code (to run at each of those join points), and Inter-type declarations are declarations that cut across classes and their hierarchies. In our case, for each component or connector, pointcuts specify time spots: whenever a port sends or receives a message; pieces of code brought together by advice with pointcuts are
the generated monitoring code; and \textit{Inter-type declaration} specifies helper variables and methods. Fig. 4, which is a part of generated aspect for composition \textit{Customer} in Fig. 2, clearly shows the way to weave runtime verification code into functionality code through aspects. Currently the SAM parser can handle future time linear temporal formulae as well as past time linear temporal formulae.

By combining runtime verification and automated implementation of software architecture, we can obtain the following benefits:

- The transformation from design models to implementations is generally informal, therefore error-prone. Automated implementation provide a means to prevent man-made errors, and runtime verification can validate transformation indirectly.
- Runtime verification at implementation level is a natural complement to analysis techniques of design level. Not all properties can be verified against a design model either due to the state space explosion problem or due to characteristic of open-systems. In either case, runtime verification can be explored to verify the correctness of design models.
- Runtime verification provides a mechanism to handle exceptions of implementations that are not detected during development or testing.

6.2 Results

The basic requirements for the online shopping system are correctness, robustness and reliability. Part of system properties we checked are follows:

1. Composition \textit{Customer} obtains a message

\[
m2 = < S_5, uid, pname, category, price >
\]

in the port \textit{response}_{15}(m2), it indicates that sometime before the customer sent out a message

\[
m1 = < S_5, uid, pReq, category >
\]

through port \textit{request}_{12}(m1). This can be expressed by the following formulae on composition \textit{Customer}:

\[
\Box(response_{15}(< S_5, uid, pname, category, price >)) \quad (1)
\]

\[
\Box(request_{12}(< S_5, uid, “pReq”, category >) \rightarrow \\
\Box(response_{15}(< S_5, uid, pname, category, price >))) (2)
\]

2. Selective response property. When a customer sends a checkout request with products (s)he wants, eventually (s)he gets either an order and shipment information or the error response on this payment process. The formula is

\[
\Box((checkout_{12}(< S_5, uid, “checkout” >) \land \\
prdLCs_{12}(< S_5, uid, pname, category, price, quantity >)) \rightarrow \Box((ordNum_{15}(< S_5, uid, ordNum >) \land \\
shipEmail_{15}(< S_5, uid, “shipEmail” >)) \lor \\
chkoutErr_{15}(< S_5, uid, “ccErr” >))) (3)
\]

3. Conjoined reverse reasoning property. This property involves more than one reasons about a result. When a customer receives an order information, we can reversely deduce that the first the customer got a response on a request for the product(s), second the customer provided his/her credit card information sometime before, finally the customer got the shipment information. The LTL formula for this property is

\[
\Box((ordNum_{15}(< S_5, uid, ordNum >) \rightarrow \\
\Box(((request_{12}(< S_5, uid, “pReq”, category >) \rightarrow \\
\Box(response_{15}(< S_5, uid, pname, category, price >)) \land \\
\Box(shipEmail_{15}(< S_5, uid, “shipEmail” >) \land \\
\Box(uInfoReq_{15}(< S_5, uid, “uReq” >) \rightarrow \\
\Box(uInfoRes_{12}(< S_5, uid, name, ccNum, limit, ccExp >)))))))) (4)
\]

4. Cause-and-effect property. This involves more than two components/compositions, and there is a timing order between each two. For instance, the E-Company request to authorize a customer’s payment information (credit card form). After component \textit{PaymentProcess} responses, the E-Company sends out either an order and shipment information or the credit card error information to the component \textit{Customer}. The LTL formula for this property is

\[
\Box((ccAuth_{23}(< S_5, uid, name, ccNum, limit, ccExp, tprice >) \rightarrow \\
\Box((ccRes_{28}(< S_5, uid, ccStatus >) \rightarrow \\
\Box((ccErr_{21}(< S_5, uid, “ccErr” >) \lor \\
(shipEmail_{15}(< S_5, uid, “shipEmail” >) \land \\
ordNum_{21}(< S_5, uid, ordNum >)))))) (5)
\]

The translation of SAM model and generated code of verification takes about 1.5 seconds for the SAM parser on a P4 2.4Ghz machine with 512MB RAM. The generated implementation has 280 files, and it is executable without any modification. Most of files are the implementation of components or connectors behavior (Petri nets).

From the log file, we also can see that the formulae 1, 3, 5 holds. However, the evaluations of formulae 2 is neither true nor false. This seems strange at first since the purpose of runtime verification was to check if formula are satisfied or not. However this result is correct because these two formulae are past time LTL, which are supposed to be always satisfied. In other words, runtime verification code only reports exception for past time LTL formulae if it is violated, just like code in Fig. 4. Therefore, the unsure result for formula 2 indicate that there are no violation detected. Same reason for the formula 4. This means these formulae hold during the program execution, which assures the behavioral correctness of \textit{Customer}, \textit{E-Company}, and \textit{PaymentProcess}.

7. \textbf{CONCLUSIONS AND FUTURE WORKS}

We have presented the basic idea of SO-SAM model. Software architecture of a web services-oriented system can be described by building an executable SO-SAM model. The behaviors of such a model can be described by a Petri net and simulated to allow advanced analysis, corrections and further refinements. With the properties specified by temporal logic formulae, the model can be validated against these properties. As a result, our modeling and validating approach helps enhance the reliability of web service-oriented applications. Furthermore, it supports the component-based concepts and object-oriented paradigm. Specification formalism in SO-SAM is component-based and with port as interface to outside it is object-oriented, and executable, expressive, comprehensive, and yet easy to use. Thus a wide body of theories available for Petri nets is available for analyzing a system design.

Currently we provide a tool that support automatically transfer the SO-SAM model to XML specifications. However, transferring the WSDL specifications into the SO-SAM specifications is still done manually. In order to monitor and verify more real-life applications using SO-SAM, the translation from WSDL into SO-SAM must be automated.

Our future work includes establishing an automation engine that translates web services written in WSDL into SO-SAM, and building SO-SAM models over real-life applications.
8. ACKNOWLEDGMENTS

This work is supported in part by NSF under grant HRD-0317692 and by NASA under grant NAG 2-1440. This work is also partially supported by Florida International University Graduate School Dissertation Fellowship.

9. REFERENCES


http://www.w3c.org/XML.


http://www.w3.org/TR/soap/.


http://www.w3.org/TR/2002/NOTE-wsccl1-20020314/.

http://www.w3.org/TR/wSDL.


