WS-Net: A Petri-net Based Specification Model for Web Services

Jia Zhang
Department of Computer Science
Northern Illinois University
DeKalb, IL 60115
jiazhang@cs.niu.edu

Carl K. Chang
Department of Computer Science
Iowa State University
Ames, IA 50011
chang@cs.iastate.edu

Jen-Yao Chung
IBM T.J. Watson Research
Yorktown Heights, New York 10598
jychung@us.ibm.com

Seong W. Kim
Samsung Advanced Institute of Technology
P.O. Box 111, Suwon 440-600 Korea
seongwoon.kim@samsung.com

Abstract

The emerging paradigm of web services opens a new way of web application design and development to quickly develop and deploy web applications by integrating independently published web services components to conduct new business transactions. As research aiming at facilitating web services integration and verification, WS-Net is an executable architectural description language incorporating the semantics of Colored Petri-net with the style and understandability of object-oriented concepts. WS-Net describes each web services component in three layers: interface net declares the services that the component provides to other components; interconnection net specifies the services that the component acquires to accomplish its mission; and interoperation net describes the internal operational behaviors of the component. As an architectural model that formalizes the architectural topology and behaviors of each web services component as well as the entire system, WS-Net facilitates the verification and monitoring of web services integration.

1. Introduction

The emerging paradigm of web services opens a new way of web application design and development to quickly develop and deploy web applications by integrating independently published web services components to conduct new business transactions. Accordingly, a web services-oriented system refers to a system that integrates multiple web services components. Just as other software systems, software architecture is critical to decide the success of a web services-oriented system. According to the American Heritage Dictionary, architecture is "the art and science of designing and erecting buildings, a structure of structures collectively, a style and method of design and construction" [4]. In other words, software architecture plays an essential role in providing the right insights, triggering the right questions, and offering general tools for thoughts. Because software architecture is normally used as a vehicle for communications between different stakeholders, the architecture specification that represents a complex web services-oriented system at a high level abstraction should capture: (1) the structural properties and web services component interactions, (2) the behavioral functionality of each major high level web services component, and (3) the behavioral functionality of the entire system. In recent years, researchers from both industry and academia have been developing a number of web services-oriented Architecture Description Languages (ADL), typically Web Services Description Language (WSDL) [17], Web Services Flow Language (WSFL) [8], Business Process Execution Language for Web Services (BPEL4WS) [6], Web Service Choreography Interface (WSCI) [13], XLANG [14], etc. However, these ADLs either merely focus on static functional descriptions of web services component as a whole, or concentrate only on the behavioral integration between web services components. Furthermore, none of these current ADLs support dynamic verification and monitoring of the integrated system.

As research aiming at facilitating web services integration, this paper presents an architectural description language called Web Service Net (WS-Net) to overcome the drawbacks of current web services-oriented ADLs. WS-Net is an executable specification language based on the Colored Petri Net (CPN) [7] semantics with the style and understandability of the object-oriented paradigm. Supporting modern software engineering philosophies equipped with component-based notations, WS-Net provides an approach to verify and monitor the dynamic integration of a web services-oriented software system. The remainder of this paper is organized as follows. In Section 2, related work is discussed. In Section 3, we review CPN for high-level design. In Section 4, we present three-layer WS-Net specification. In Section 5, we draw conclusions and describe future work.

2. Related Work

In recent years, researchers have been doing much work on modeling web services-oriented system architecture. Generally, all of the proliferating work is built upon eXtensible Markup Language (XML) [18] technology. Among them, Web Services Description Language (WSDL) [17] is the basis of other work. Intending to formally and precisely define a web service, WSDL from W3C (**http://www.w3c.org**) is becoming the *ad hoc* standard for web service publication. However, WSDL can only specify limited information of a web service, such as the function names and limited input and output information [3]. In recognition of this problem, researchers from both academia and industry have been developing other description languages to extend the power of WSDL to depict system architecture. The following are some outstanding examples:

- Web Services Flow Language (WSFL) [8] is a WSDL-based language focusing on describing the interactions between web services components. WSFL defines the interaction pattern of a collection of web services, as well as the usage pattern of a collection of web services in order to achieve a specific business goal.
- The Business Process Execution Language for Web Services (BPEL4WS) [6] specifies an interoperable integration model aiming at facilitating the automatic integration of web services components. BPEL4WS formally defines a business process and process integration.
- Web Service Choreography Interface (WSCI) [13] utilizes the flow of messages to define the relationship and interactions between web services components. According to WSCI, a web services component has both a static interface and a dynamic interface when it participates in a message exchange with other web services components.
- XLANG [14] considers the behavior of a system as an aggregation of the behavior of each web services component. Therefore, XLANG specifies the behavior of each web services component independently. The interactions between web services are conducted via message passing, which is expressed as the operations in WSDL.
- Web Service Capability Description Language (SCDL) is built upon the method of an abstract finite-state machine, aiming at precisely describing, advertising, requesting, and matching web service capabilities. Defining four types of atomic web service capability matches as exact match, plug-in match, relaxed match, and not relevant, SCDL provides a theoretical basis to define web service capability matching.

However, these ADLs 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. SCDL and its previous version SDL [5] are still at early development stages [16]. Therefore, the usage of SCDL in web service applications is still unclear. Furthermore, none of these current ADLs support dynamic verification and monitoring of the system integrated.

3. Petri Nets for High-Level Architectural Design

Petri nets [7] is a well-known and tested abstract and formal model of data and control flow in a system that exhibits synchronous and asynchronous behaviors. With a strong theoretical foundation, the mature hierarchical Petri nets is able to manage the complexity of large-scale systems, and possesses powerful analysis capability. Colored Petri Net (CPN) [7] extends the Petri nets to model both the static and dynamic properties of a system. The graphical part of CPN depicts the static architectural structure of a system. Combined with other powerful elements such as colored tokens and simulation rules, CPN is very powerful in modeling dynamic behaviors of a system. Earlier researchers have conducted a large amount of work to utilize CPN to model the system architecture. EDDA [15] combines Petri nets and SADT technology for high-level system specifications. Although EDDA successfully combines the semantics of Petri-nets with the syntax of SADT, it lacks the ability to specify modern software systems, as EDDA does not embody the object-oriented paradigm and the component-engineering concept [1]. Pinci and Shapiro present an automatic mechanism to translate SADT diagrams into Hierarchical CP-nets (HCP-nets), and in turn to convert HCP-nets into Standard ML executable code [9]. This similar SADT-like Petri-net based system specification suffers the same problems as faced by EDDA due to the rigid structural nature of SADT and its lack of object-oriented concepts.

Our previous work released an I^3 [1] layered executable architectural model. However, I^3 is based upon the Structural Analysis and Design Technology (SADT) [10], which is a traditional functional decomposition and data flow-centered methodology. Here we aim at integrating CPN with the style and understandability of the object-oriented paradigm. In addition, I^3 intends to present a generic specification model oriented to generic component-based software systems. Our work reported here focuses on web services-oriented system architecture and seamlessly integrates with WSDL and XML

technology.

In summary, although our work was strongly influenced by EDDA and I[3], we have enhanced the state of the art by supporting modern software engineering philosophies equipped with object-oriented and component-based notations and applied to web services-oriented systems, as well as integrated with WSDL and XML.

4. WS-Net Specification

In order to specify both the static and dynamic architectural features of a web services-oriented system, we present Web Service Net (WS-Net) as an executable architectural specification language. WS-Net specifies a web services-oriented software architecture as a set of connected architectural components described as nets. The architectural components correspond to functional units in the system, and one architectural component may in turn be composed of several smaller architectural components. The entire system can be viewed as a highest-level architectural 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 *ad hoc* transportation standard in the realm of web services. The message passing mediates the interactions between architectural components via the rules that regulate the component interactions. In our model, we will use the concept of connector [11] in CPN to model the message passing.

WS-Net defines each architectural component in a three-layer specification: interface net, interconnection net, and interoperation net. The interface net declares the services to be provided by each web service component. The interconnection net specifies the web services to be acquired from other web service components to accomplish its own mission. The interoperation net describes the functionality of each web services component and the entire system in terms of data flow. Each component has an interface net definition and it is accessed only via the interface. The definition of the interface net follows that of WSDL. The interconnection net specifies the operations to be acquired from other components to perform its execution. In the Interconnection net, each operation required is further specified by a set of foreign transitions, which represents the operations of other components. Therefore, the interface net identifies each component in the system as a unique functional object, and the interconnection net specifies the relationships between components. As a result, we can visualize the entire topological view of a system by interconnecting each of the interconnection nets

according to our unique component-interconnection technique, which will be discussed in the later sections. The interoperation net describes the dynamic behaviors of a component by focusing on its internal operational nature. The goal of the interoperation net is to dissect each operation into fundamental process units, which, taken together, define the required functional content of the operation. Each transition representing an operation of the component will be decomposed into sub-transitions representing fundamental process units. Control flow and data flow are used for describing intercommunication between process units.

A simplified user commit model is used as an example to elucidate the fundamental idea of WS-Net throughout this paper. This example illuminates a system whose service is to provide a channel for users to communicate with a central storage, that is, to commit data to the storage and retrieve data from the storage. Three distributed web services components are identified in the system: a queue, a commit engine, and an integrator. The queue component is the central storage place that accepts user commitment and provides information requested; the commit engine assists users to commit requests to the queue; and the integrator conducts continuous integration from the queue. We assume that different components interact with each other via SOAP.

4.1 Interface Net

Constructing a WS-Net specification starts by identifying the architectural components from the system specification. The interface net defines expected responsibilities, or features, of each architectural component by specifying the interface of a set of semantically related operations provided by a component. The interface here denotes the name of the service provided and the signature information in order to invoke the service. An architectural component can be accessed only through its interface. In the interface net, each service is modeled as a transition of Petri net. Therefore a transition is called a service transition. The name of the service transition refers to the service to be provided by the component. Each service transition has an input place called input port, where the service received the invocation, and an output place called output port, where the result of a service is placed before being returned to the service caller. The interface net can be implemented via WSDL.

A WS-Net specification of an architectural component can be denoted as $C_i \in C$, where C is the set of all web services components identified in the software architecture. The interface net of the component C_i can be informally represented as:

$$C_i = \bigcup_j S_{ij}, \; S_{ij} \in S_i$$

where S_i is a set of services provided by C_i. Each service $S_{ij} \in S_i$ is represented by a tuple

$$S_{ij} = (PI_{ij}, PO_{ij}, T_{ij}, A_{ij}, c),$$

where PI_{ij} and PO_{ij} are the input port and the output port of S_{ij}, respectively; T_{ij} represents the service S_{ij} as a transition; A_{ij} represents the input and out arcs for the transition T_{ij}, and the color inscriptions of the places represent the signature information of the operation as in CPN; and, c is a color function for the places.

The goal of the interface net is to define the services that must be provided by the component. The names attached to the service transition inscription represent the names of the services. As in Design/CPN [2], names of the places and transitions are the labels to identify the places and transitions, and they are not considered as semantic inscriptions of Petri net. In Colored Petri Nets [2,7], they are used to help designers understand the specification and to support hierarchical composition of pages, such as transition substitution and place fusion. The signature information of each service can be described by color inscription on input and output ports (places). However, it is unimportant to specify the detailed data structure at this stage of the design. The main purpose of coloring places is to help people understand the usage of a component at the architectural level. Therefore, the only imperative information here is what kind of information needs to be provided to invoke the service of the component. The interface net specification of the *queue* component is illustrated in Figure 1.

As shown in Figure 1, *queue* provides two services *enqueue* and *dequeue*, which accept user commitment and reply to user requests respectively. In order for the services to be invoked, the input ports of the services must receive proper tokens. *Enqueue* receives an *item* token and returns a Boolean *result* as either success or fail; *Dequeue* receives a *request* as a unit token and returns a *dq_rslt* in case of success, or *err* in case of failure caused by empty buffer. The unit color set has no predefined service that uses it, but is very useful as a placeholder [7].

In addition to the inscription for places, transitions, and arcs as used in Colored Petri Nets, WS-Net provides additional inscriptions for components and each service in the components. For the component inscription, component name is provided to identify the component. For a service inscription, service name and connector type information are provided. Connector type implies the possible protocols to be used when the service is invoked. Since multiple protocols are commonly used, connector type can have multiple entries. In our example, both *enqueue* and *dequeue* services are invoked via the SOAP protocol. Similar to the name inscription for places and transitions, these inscriptions for components and services
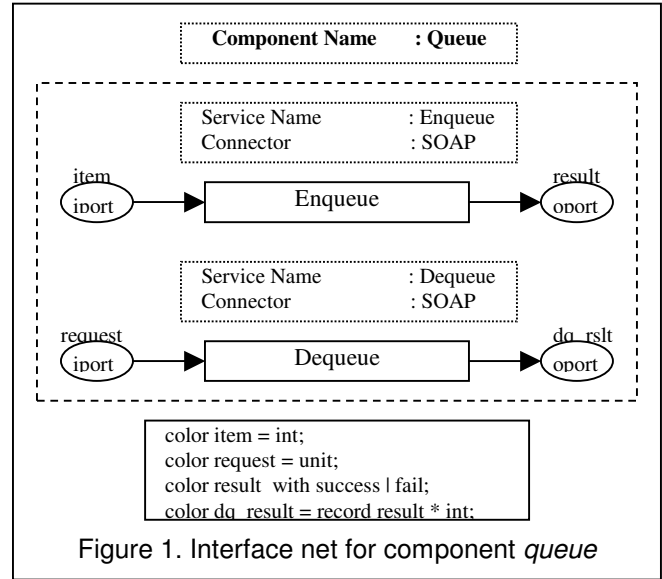

Figure 1. Interface net for component *queue*

are not considered as Petri-Net inscriptions. These inscriptions will be used to interconnect architectural components.

In our example, the *commit engine* and the *integrator* components will use the *enqueue* and *dequeue* services from the *queue* component. Therefore, the *commit engine* and *integrator* will be client components for the *queue* component. Figure 2 shows the interface net of the *commit engine*, which provides one service called *commit*. The *commit engine* will send some *data* as parameters to the *queue* and receive an acknowledgement *msg*. Similarly, as shown in Figure 3, the *integrator* provides one service called *retrieve,* and will send a request *rqst* to the *queue* and receive some *data* as reply. As we explained previously, the color of token is used here. We assume that the SOAP protocol will be used when the *commit engine* and the *integrator* components communicate with the *queue*. Since an interface net specifies only the services provided by the component, it does not specify connections between the components.
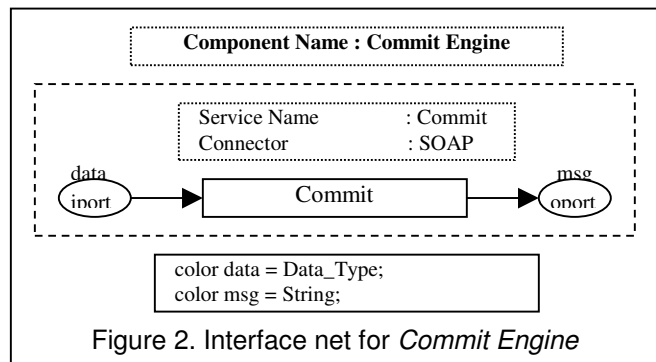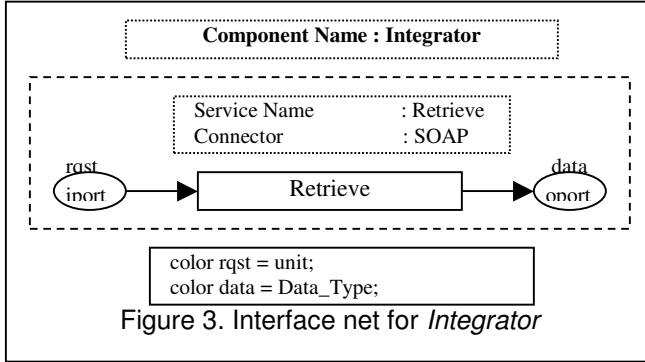
4.2 Interconnection Net

Figure 2. Interface net for *Commit Engine*

Component Name : Integrator

Service Name	: Retrieve
Connector	: SOAP

rqst → inort → **Retrieve** → onort → data

color rqst = unit;
color data = Data_Type;

Figure 3. Interface net for *Integrator*

In order to describe the relationships between architectural components, we need to specify both the provided services (i.e. via interface net specification) and required services of the components. By specifying all the required services of the components, the interconnection net describes all the possible dependencies upon other components. The interconnection net is not mandatory however; instead, it is imperative only when a component requires foreign services to perform its own commission. For instance, in our example, the *queue* component does not require any other services as support; therefore, there will be no interconnection net associated with the *queue* component. The interconnection net depicts a client/server relationship between components. If component C_i requests a service from component C_j, C_i is called the client component, and C_j is called a server component. Therefore, a component can act as a client component sometimes, and as a server component at other times.

WS-Net chooses to define the required services as foreign services since the services need to be performed by other components. Conforming to the definitions in CPN, in the interconnection net, the required services are specified as a special kind of transition called a foreign transition. As in the interface net, the interconnection net specifies an architectural component as a set of provided services. Each provided service containing foreign transitions are in turn decomposed into a set of required foreign services. A service S_{ij} requiring foreign services is represented as a tuple

$S_{ij} = (PI_{ij}, PO_{ij}, QI_{ij}, QO_{ij}, TS_{ij}, TE_{ij}, TF_{ij}, A_{ij}, c)$,

where PI_{ij} and PO_{ij} are the input port and the output port for S_{ij} respectively, as in the interface net. TS_{ij} and TE_{ij} represent the start and end of the service S_{ij}; therefore, the input place of TS_{ij} is always the input port PI_{ij}, and the output place of TE_{ij} is always the output port PO_{ij}. TF_{ij} is a set of foreign transitions. A foreign transition is an abstract view of the service provided by the server component. The input and output places of a foreign transition are called input sockets and output sockets, respectively. QI_{ij} is a set of input sockets, and QO_{ij} is a set of output sockets. A_{ij} is a set of input and output arcs for the transitions. As in the interface net, c represents a color

function.

A component may require multiple foreign services before it can perform its execution. Figure 4 illustrates a service that requires n foreign transitions. For services that do not require any foreign transitions, the same service specification of the interface net will suffice. Foreign transitions also have inscriptions similar to the provided services of the component. However, as shown in Figure 4, inscriptions of foreign transitions contain names of the server components, names of the services required from the server components, and the type of connector to be used to invoke each foreign transition. These service names and component names are used to identify the services of server components represented by foreign transitions. In addition to the inscriptions, the color set of the sockets of the foreign services in the client component and its corresponding color set for the services of the server component must be compatible.

Figure 5 and Figure 6 show the interconnection nets of the component *commit engine* and *integrator,* respectively. The *commit* service needs to invoke the *enqueue* service of the component *queue,* and the *retrieve* service needs to invoke the *dequeue* service. Therefore, *enqueue* and *dequeue* services are represented as foreign services. Inscriptions for the foreign transitions show that they are calling *enqueue* and *dequeue* services from the component *queue* via SOAP.

After specifying individual components in terms of the interface nets and the interconnection nets, we are ready to visualize the entire topological view of the system by interconnecting all of these WS-Net components. Firing a foreign transition means executing the corresponding

Figure 4. A service with n foreign transitions

Figure 5. Interconnection net for *commit engine*
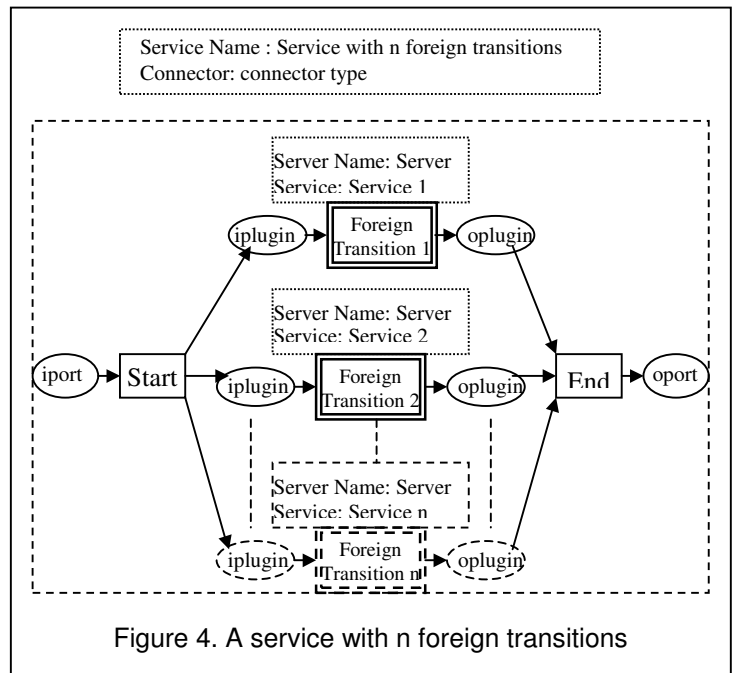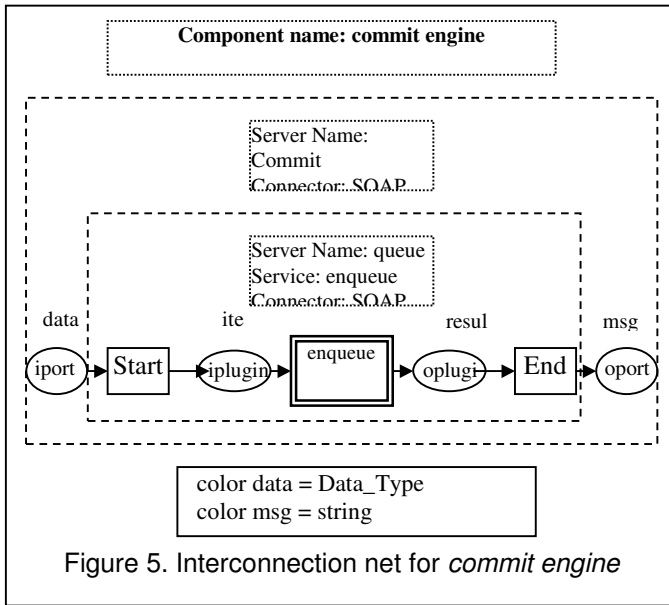
service transition of the server component. Therefore, connecting WS-Net components can be achieved by merging the ports of the client components with the ports of the server component, after removing foreign transitions from the client components. In our WS-Net, a special kind of transitions is used to connect ports. This transition is called a connector transition, and it is named by a connector type. Figure 7 shows the connected interconnection nets that describe the entire information-communication model by interconnecting the *commit engine* and the *integrator* with the *queue* using SOAP connectors.

4.3 Interoperation Net

Figure 6. Interconnection net for *integrator*
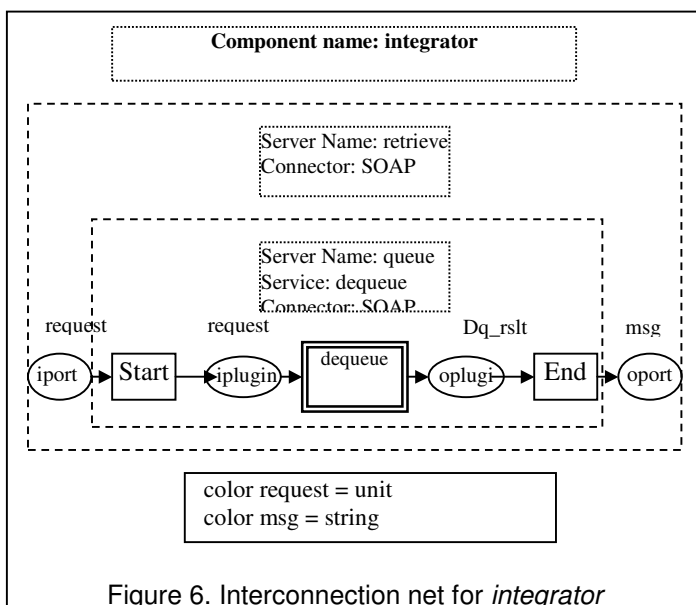
The interoperation net describes the dynamic behaviors of a component by focusing on the operational nature of the component. The goal of the interoperation net is to dissect each service into fundamental process units which, taken together, define the required functional contents of the service. This is similar to the SADT functional decomposition, where each transition representing the operations of the component to be decomposed into sub-transitions to represent fundamental operational state. One of the most important differences between the decomposition in our interoperation net and SADT is that, the interoperation net uses the decomposition as a means of expressing the behaviors of the services provided by an architectural component, rather than functional decomposition for modularization used in SADT. As in SADT, the control and data flow are used to describe interactions between process units. It is very important to distinguish foreign transitions from the detailed processes. These foreign transitions along with plugin places will be used to interconnect the interoperation nets to form the entire system view. Like other Petri net-based high level design representations, places are used to represent the control or data; and transitions are used to represent the processes.

Our previous research reveals that the straightforward techniques converting functional data flow to Petri nets have a potential problem in repeated (persistent) simulations of the nets [1]. To solve this problem, our WS-Net distinguishes the persistent data from the


Figure 7. Unfolding Interconnection net
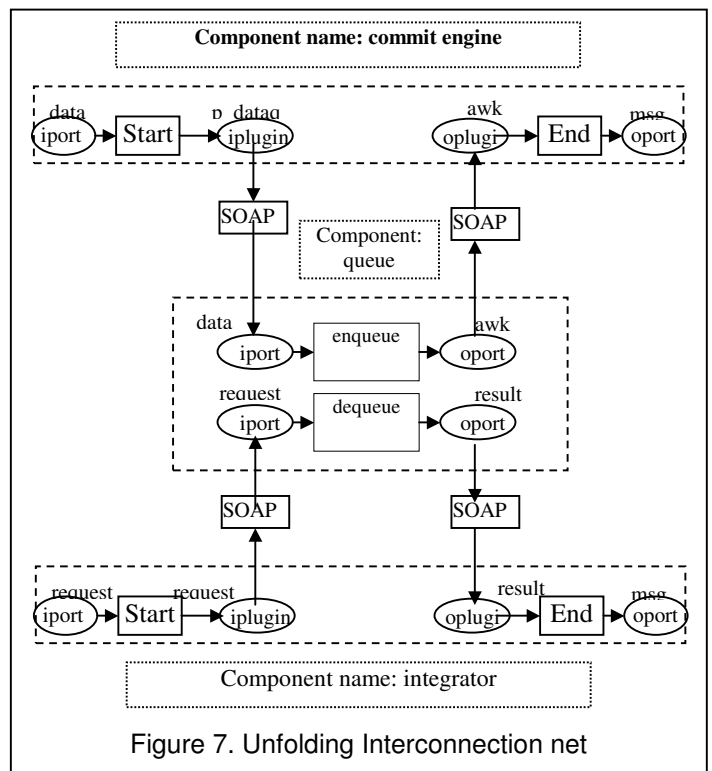
transient data. Persistent data items are similar to the data attributes of a class in the object-oriented paradigm. These persistent data items represent the state of the component, and they should exist throughout the lifetime of the component. On the other hand, transient data items are produced by one process and will be immediately consumed by another process. Therefore, transient data items are created only when they are needed and destroyed upon the completion of the service. Since all transient tokens are created by the local transitions and all persistent tokens are restored before the completion of the service, repeated simulation of the net is possible. In converting functional data flow models to Petri nets, we also face the concurrency and choice problems [15]. Those problems need to be addressed properly by a system engineer who builds the system architecture by using WS-Net.

A service $S_{ij} \in S_i$ of component C_i is represented as follows:

$S_{ij} = (PI_{ij}, PO_{ij}, PT_{ij}, PP_{ij}, QI_{ij}, QO_{ij}, TL_{ij}, TF_{ij}, A_{ij}, c, G, E, IN)$,

where PI_{ij} and PO_{ij} are the input ports, and PT_{ij} and PP_{ij}, are a set of transient data places and a set of persistent data places respectively. Persistent places are represented as boldface circles. TL_{ij} is a set of local transitions; and TF_{ij} is a set of foreign transitions. QI_{ij} is a set of input socket places serving as input places for the foreign transitions,;and QO_{ij} is a set of output socket places serving as output places for the foreign transitions. A_{ij} is a set of input and output arcs of the transitions. To describe the functional behaviors of a component, we can use all the inscriptions used in Colored Petri Net [7]. As before, c is a color function to represent the color sets for the places. G is a guard function for the transitions. E is an arc expression function, and IN is an initialization function for the tokens.

In our example, the *queue* component has *enqueue* and *dequeue* services. The control and data are represented by places; and processes are represented by transitions. Figure 8 shows the first phase of the interoperation description of the *queue* component. *Count* and *storage* places are defined as persistent data and represented with boldface circles. Since the persistent data may exist throughout the lifetime of a component, we need to initialize the persistent places with proper tokens for later simulations. Tokens in the transient places will be produced as a result of firing transitions. It is very common for persistent data items to be shared by other services in the same component. If different services use the same persistent data, they need to be merged using the place-fusion technique in high level Petri nets. As shown in Figure 8, *count* and *storage* are persistent places of both the *enqueue* and *dequeue* services. By merging those persistent places of two services, the interoperation net for the *queue* component can be completed.

Figure 8. First phase of Interoperation net for *queue*
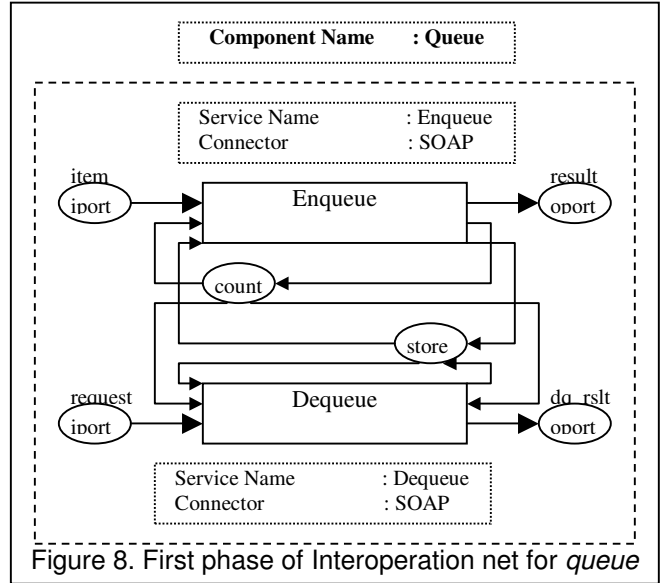
As we further decompose the functional behaviors of each service, we can get a further complex interoperation net. Figure 9 shows a more detailed interoperation net of the service *enqueue* of the *queue* component. After all the interoperation nets of the architectural components are specified, we can again visualize the entire system topology by connecting plugins of the client components with the ports of the server components using connector transitions. WS-Net provides an interconnection mechanism across different levels of component diagrams. Interconnections can be visualized by: (1) interoperation nets of sender and receiver components, and (2) the interface net of the sender, receiver, and channel components. We believe that this is a very important feature to visualize very large systems. By applying such visual abstractions, such as replacing large interoperation nets with simpler interconnection nets or even with interface nets, complicated nets can be effectively visualized at various levels of abstraction.

Connected interoperation nets can be executed under different input scenarios to simulate the behaviors of a system. The execution proceeds by assigning initial tokens to the input ports. The execution traces need to be visualized to analyze the runtime quality attributes and to enhance communication with the user communities by providing an executable model of the system early in the development process.

5. Conclusions and Future Work

We have presented the basic idea of WS-Net. Software architecture of a web services-oriented system can be described by building an executable WS-Net model. The behaviors of such a model can be simulated to allow corrections and further refinements. As a result, our

Figure 9. Interoperation net of *enqueue*
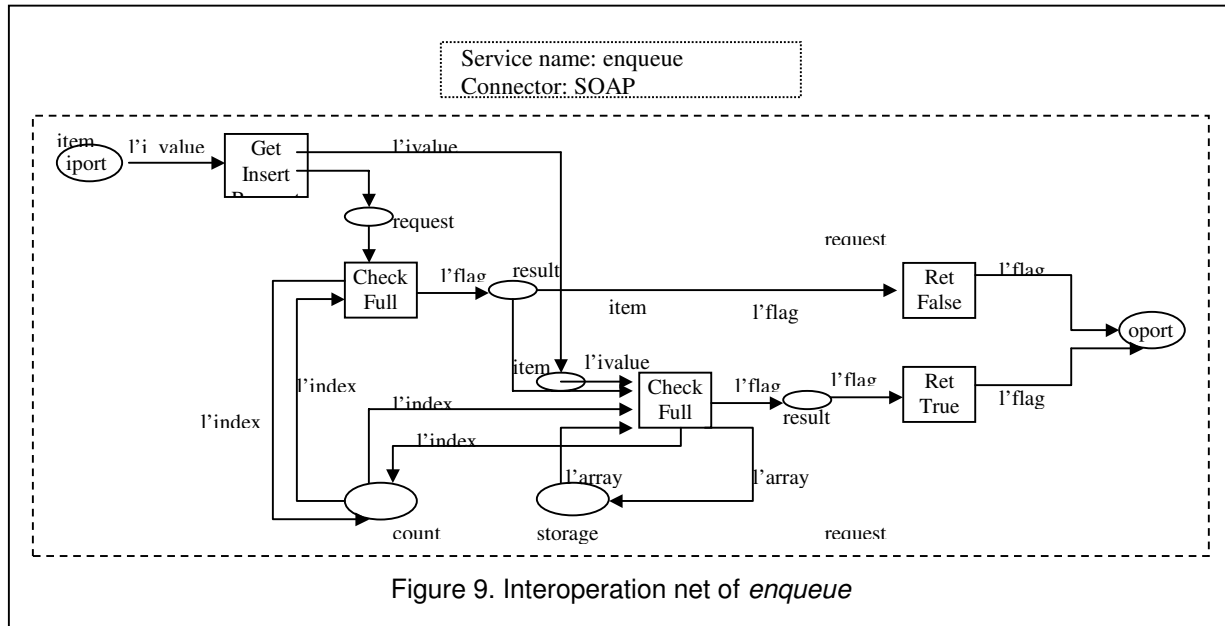
approach helps enhance the reliability of web services-oriented applications. Furthermore, it supports the object-oriented paradigm and component-based concepts. Specification formalism in WS-Net is object-oriented, executable, expressive, comprehensive, and yet easy to use. Thus a wide body of theories available for Petri nets is available for analyzing a system design.

However, manually transferring the WSDL specifications into the WS-Net specifications is not a trivial job. That is why currently we have only built some simple experiments, e.g. the example described in the paper. In order for our WS-Net to monitor and verify real-life applications, the translation from WSDL into WS-Net must be automated.

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

6. References

[1] C.K. Chang and S. Kim, "I^3: A Petri-Net Based Specification Method for Architectural Components", *Proceedings of IEEE 23th Annual International Computer Software and Applications Conference (COMPSAC'99),* Phoenix, AR, USA, Oct. 25-26, 1999, pp. 396-402.

[2] Design/CPN Reference Manual for X-Windows Version 2.0, Meta Software Corporation, 1993.

[3] X. Gao, J. Yang, and M.P. Papazoglou, "The Capability Matching of Web Services", *Proceedings of the IEEE International Symposium on Multimedia Software Engineering (MSE'02)*, Newport Beach, CA, USA, Dec. 11-13, 2002, pp. 56-63.

[4] *The American Heritage Dictionary of the English Language*, 4th edition, Houghton Mifflin Company, 2000.

[5] W.J. van den Heuvel, J. Yang, and M. Papazoglou, "Service Representation, Discovery and Composition for E-marketplaces", *Proceedings of the 6th International Conference on Cooperative Information Systems (COOPIS'01)*, Trento, Italy, Sep. 5-7, 2001.

[6] IBM Corporation, "Business Process Execution Language for Web Services (BPEL4WS)," Version 1.0, 2002.

[7] K. Jensen, "Coloured Petri Nets: A High Level language for System Design and Analysis", *Lecture Notes in Computer Science, Advances in Petri Nets*, 1990.

[8] F. Leymann, "Web Services Flow Language (WSFL 1.0)," IBM Corporation, 2001.

[9] V. Pinci and R. Shapiro, "An Integrated Software Development Methodology based on Hierarchical Colored Petri Nets", *Lecture Nets in Computer Science, Advances in Petri Nets,* 1990, pp. 227-252.

[10] D. Ross, "Application and extensions of SADT", *IEEE Computer*, 18*(4)*, Apr. 1984, pp. 25-35.

[11] M. Shaw, R. DeLine, D.V. Klein, T.L. Ross, D.M. Young, and G. Zelesnik, "Abstractions for Software Architecture and Tools to Support Them", *IEEE Transactions on Software Engineering,* Special Issue on Software Architecture, 21*(4)*, Apr. 1995.

[12] http://www.w3.org/TR/SOAP, accessed on Oct. 25, 2003.

[13] Sun Microsystems, "Web Service Choreography Interface (WSCI)," Version 1.0, 2002.

[14] S. Thatte, "XLANG - Web Services for Business Process Design," Microsoft Corporation, 2001.

[15] W. Trattnig and H. Kerner, "EDDA-A Very High-level Programming and Specification Language in the Style of SADT", *Proceedings of IEEE Annual InternationalComputer Software and Applications Conference (COMPSAC'80)*, Oct. 1980, pp. 436-443.

[16] A. Tsalgatidou and T. Pilioura, "An Overview of Standards and Related Technology in Web Services", *Distributed and Parallel Databases,* 12, 2002, pp. 135-162.

[17] http://www.w3.org/TR/wsdl, accessed on Oct. 25, 2003.

[18] .B. McLaughlin and M. Loukides, *Java and XML*, O'Reilly Java Tools, 2001.