

Model Driven Development of a Service Oriented Architecture (SOA) Using Colored Petri Nets

Vijay Gehlot¹, Thomas Way¹, Robert Beck¹, and Peter DePasquale²

¹ Center of Excellence in Enterprise Technology, Department of Computing Sciences
Villanova University, Villanova, Pennsylvania 19085, USA
{vijay.gehlot, thomas.way, robert.beck}@villanova.edu
<http://ceet.villanova.edu>

² Department of Computer Science, The College of New Jersey, Ewing, NJ 08628, USA
depasqua@tcnj.edu

Abstract. Service-Oriented Architecture (SOA) is achieving widespread acceptance in a variety of enterprise systems, due to its inherent flexibility and interoperability, improving upon the more tradition and less supportable “stovepipe” approach. The high degree of concurrency and both synchronous and asynchronous communications inherent in SOA makes it a good candidate for a Petri Nets based model driven development (MDD). Such an approach, with its underlying verification and validation implications, becomes more crucial in mission-critical applications, such as those with defense implications. This paper reports on our experience with using Colored Petri Nets (CPNs) for model driven development and quality assessment of a defense-targeted service-oriented software architecture. We identify features of CPN that have resulted in ease of adoption as a modeling tool in our present setting. Preliminary results are provided which support the use of CPNs as a basis for model driven software development, and verification and validation (V&V) for quality assurance of highly concurrent and mission-critical SOAs.

1 Introduction

Designers of enterprise architectures have embraced the Service Oriented Architecture (SOA) approach, which leverages significant advances in distributed computing and networking technologies to enable large scale interoperability [1,2]. Although the SOA approach promotes flexibility, reuse and decoupling of functionality from implementation, the inherent complexity of the enterprise class of services makes the verification and validation (V&V) of such systems difficult [3]. Specific requirements of SOA applications to net-centric Department of Defense (DoD) deployments, such as stringent service guarantees, fault tolerance and security, among others [4], coupled with the significant costs involved in fulfilling these strict requirements, suggests a need for a model driven development and quality assurance approach that can accommodate the highly concurrent nature of enterprise uses of SOA. We show how

support for hierarchical and abstraction features, concurrency, and both synchronous and asynchronous communications in Colored Petri Nets (CPNs) enable modeling real-world SOA implementations to perform V&V and quality assurance required for DoD deployments. As part of a model driven approach, the created model is also intended to be used for quality predictions.

2 Service Oriented Architectures

Service-Oriented architecture (SOA) is a distributed network architecture design approach that separates services provided from the entities that consume those services. Services communicate with each other, yet are self-contained and do not depend on the state of other services, leading to a loosely coupled architecture that, as a result of this decoupling of services, is easily reconfigurable. Generally, SOA is defined as an “enterprise-wide IT architecture that promotes loose coupling, reuse, and interoperability between systems,” with the more specific view as “architectures making use of Web service technologies such as SOAP, WSDL, and UDDI... conforming to the W3C Web services architecture (WSA).” [2]

The SOA approach is attractive for enterprise systems because of its inherent flexibility and reusability and its isolation of functionality from the details of implementation. Developers of SOA providers and consumers design complex software systems using implementation-neutral interfaces, rather than less flexible, highly integrated interfaces resulting from proprietary specification and design approaches. By maintaining interoperability at the interface, developers evolve services with isolated internal implementations of which external services need not be aware. [4] The roles described in SOA are service provider, service consumer and service discovery (Fig. 1).



Fig. 1. Overview of roles in Service-Oriented Architecture (SOA). [4]

The service provider is responsible for producing a service, making it available to service consumers by publishing a service interface in a service registry. The service

consumer makes use of the service produced by the service provider based on the rules specified in the published interface. The service discovery component of SOA provides the service publication mechanism so that service providers can make known their service interface to service consumers.

The service interfaces published by service providers adhere to the SOA approach by decoupling implementation from definition, maintaining strict configuration management so that service consumers can seamlessly migrate from one version of a service interface to another, providing backward compatibility with existing interface versions, and allowing interface and implementation versions to evolve independently. [4]

2.1 DoD Net-Centric Enterprise Solutions for Interoperability (NESI)

The flexibility and interoperability of SOA makes it an attractive solution for enterprise services within DoD deployments. The DoD and Defense Information Systems Agency (DISA) have defined a core set of enterprise services for use in defense-related SOA systems [5], as part of an initiative called Net-Centric Enterprise Solutions for Interoperability (NESI). These Net-Centric Enterprise Services (NCES) define by NESI are a set of net-centric services, nodes and utilities for use in DoD domain- and mission-related enterprise information systems (Fig. 2, [4]). Development of NCES is ongoing with significant efforts currently underway in systems for various defense applications.

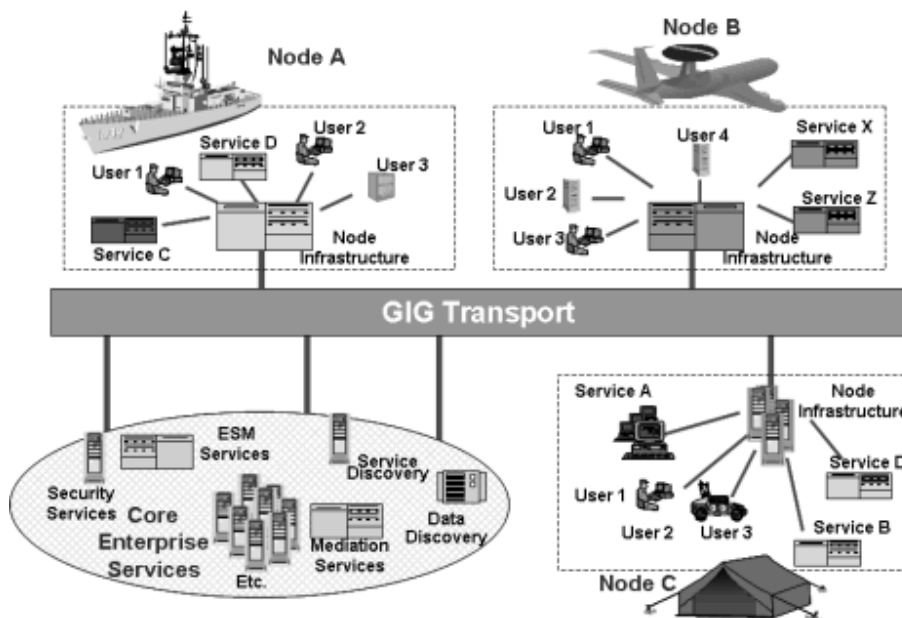


Fig. 2. Node interoperability in NESI Net-Centric Enterprise Architectures. [4]

Within SOA deployments, providers and consumers of services, when related, are collectively referred to as nodes. As illustrated in **Fig. 2**, the DoD GIG architecture describes a node as a set of information systems that form a single element in a net-centric enterprise. These nodes can include servers for web, portal, applications, and databases to provide services. To support robustness, when a node loses enterprise connectivity it should continue to serve local consumers of its services.

3 A SOA Architecture for DoD Applications

Various characteristics of SOA can be identified as necessary characteristics for broad defense applications that are otherwise not part of a traditional web-services based view of SOA. These are as follows:

- Service guarantees
- Fault tolerance
- Dynamic service discovery
- Interoperable multiple connection types
- Availability awareness
- Load balancing
- Security

A proposed general solution is to have an architecture with a mediator responsible for dynamic discovery, awareness, and load balancing. To allow interoperable multiple connection types would require a well-defined internal format and protocol with well-defined external to internal interfaces and mechanisms for internal transport and buffering. We call this architecture Service Oriented Defense Architecture (SODA).

A reference implementation of this architecture is under development by a defense contractor. Our focus is to integrate a model based approach into this software development that can be used to guide the implementation and to assess the reliability, scalability and performance of the SODA product using simulation, verification, and validation. In addition, the research goal is to provide, through modeling and analysis, feedback to the development and DoD communities to enhance their understanding of capabilities and limitations, influence architectural and technical decision making process, and set the expectations for network-centric technology architecture behaviors. The specific goals for our project are as follows:

- Gain greater understanding of the performance characteristics of multi-channel service oriented architectures.
- Achieve greater acceptance of the real world “deployability” and reliability of the multi-channel service oriented architecture, thereby accelerating “real world” legacy migrations to service-based infrastructures.
- Provide a deeper understanding of the tradeoffs that exist between performance and agility in a service-enabled environment.
- Construct a reusable set of models for researching the behaviors of large-scale deployments of service-enabled systems and the technologies that support them.

- Establish some level of benchmarking for large-scale distributed systems based on the proposed architecture.
- Create a model-based verification and validation framework for distributed systems that are based on SODA.

We discuss some details of our proposed architecture in Section 5 in conjunction with the formal model of the architecture that is being developed. We are using Colored Petri Nets (CPNs) as our modeling language. The next section gives some details of CPN.

4 Colored Petri Nets and CPN Tool

Our modeling approach is based on Colored Petri Nets (CPNs) [6]. Petri Nets provide a modeling language (or notation) well suited for distributed systems in which communication, synchronization and resource sharing play important roles. CPNs combine the strengths of ordinary Petri nets [7,8] with the strengths of a high-level programming language together with a rigorous abstraction mechanism. Petri nets provide the primitives for process interaction, while the programming language provides the primitives for the definition of data types and the manipulations of data values.

As with Petri nets, CPNs have a formal mathematical definition and a well-defined syntax and semantics. This formalization is the foundation for the different behavioral properties and the analysis methods. The complete formal definition of a CPN is given below and more details can be found in [8,9]. It should be noted that the purpose of this definition is to give a mathematically sound and unambiguous description of a CPN. In practice, however, one would create a CPN model using a tool such as CPN Tool [10]. This tool is a graphical tool that allows one to create a visual representation of a CPN model and analyze it.

Definition: A **Colored Petri Net** is a nine-tuple $(\Sigma, P, T, A, N, C, G, E, I)$, where:

- (i) Σ is a finite set of non-empty **types**, also called color sets. In the associated CPN Tool, these are described using the language CPN-ML. A **token** is a value belonging to a type.
- (ii) P is a finite set of **places**. In the associated CPN Tool these are depicted as ovals/circles.
- (iii) T is a finite set of **transitions**. In the associated CPN Tool these are depicted as rectangles.
- (iv) A is a finite set of **arcs**. In the associated CPN Tool these are depicted as directed edges. The sets of places, transitions, and arcs are pairwise disjoint, that is

$$P \cap T = P \cap A = T \cap A = \emptyset.$$
- (v) N is a **node** function. It is defined from A into $P \times T \cup T \times P$. In the associated CPN Tool this depicts the source and sink of the directed edge.

- (vi) C is a **color** function. It is defined from P into Σ .
- (vii) G is a **guard** function. It is defined from T into expressions such that:

$$\forall t \in T: [\text{Type}(G(t)) = \text{Boolean} \wedge \text{Type}(\text{Var}(G(t))) \subseteq \Sigma].$$
- (viii) E is an **arc expression** function. It is defined from A into expressions such that:

$$\forall a \in A: [\text{Type}(E(a)) = C(p)_{MS} \wedge \text{Type}(\text{Var}(E(a))) \subseteq \Sigma]$$
where p is the place of N(a) and $C(p)_{MS}$ denotes the multi-set type over the base type C(p).
- (ix) I is an **initialization** function. It is defined from P into closed expressions such that:

$$\forall p \in P: [\text{Type}(I(p)) = C(p)_{MS}].$$
In the CPN Tool this is represented as initial marking next to the associated place.

The distribution of tokens, called **marking**, in the places of a CPN determine the state of a system being modeled. The dynamic behavior of a CPN is described in terms of the **firing** of transitions. The firing of a transition takes the system from one state to another. A transition is **enabled** if the associated arc expressions of all incoming arcs can be evaluated to a multi-set, compatible with the current tokens in their respective input places, and its guard is satisfied. An enabled transition may fire by removing tokens from input places specified by the arc expression of all the incoming arcs and depositing tokens in output places specified by the arc expressions of outgoing arcs.

CPN models can be made with or without explicit reference to time. Untimed CPN models are usually used to validate the functional/logical correctness of a system, while timed CPN models are used to evaluate the performance of the system.

The time concept in CPN is based on a global clock. The clock value represents the model time. In the timed version, each token carries a time stamp. The time stamp of a token determines the earliest (simulation) time at which the token will become available.

One aspect of CPN that is attractive for creating models of large systems is being able to create **hierarchical** CPN. Hierarchical CPN allow one to relate a transition (and its surrounding arcs and places) to a separate sub-net (called a *subpage* in CPN parlance) structure. The subnet then represents the detailed description of the activity represented by the associated transition. This allows one to build a model either in top-down or bottom up manner and also allows one to either hide or expose details as necessary. Complete detail of a hierarchical CPN and its semantics can be found in [6].

5 CPN Model of SODA

We present some details of the SODA by discussing its CPN model. At the most abstract level, a general SOA, and hence SODA, can be viewed as consisting of requests for services that are sent through some discovery/mediation/transport mecha-

nism to be processed and responses returned by providers of services. In the CPN model this is the top-level page called Top and is shown in **Fig. 3**.

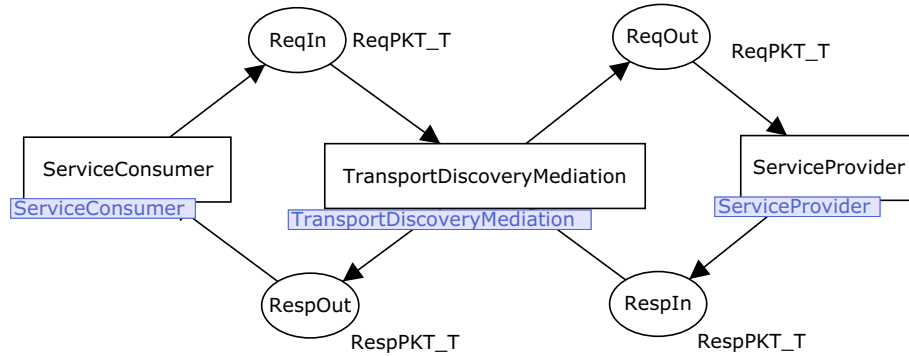


Fig. 3. Top level page in CPN model giving the most abstract view of the system.

The transport, discovery and mediation mechanism itself consists of several components. These are detailed on the CPN subpage *TransportDiscoverMediation* and are shown in **Fig. 4**. Tracing the flow of data through this net, an incoming request packet arrives via the component *Ext2IntInbound*. This component is responsible for accepting the request and possibly converting it into a desired internal format. This component is also responsible for any encryption/decryption that needs to happen as part of the request.

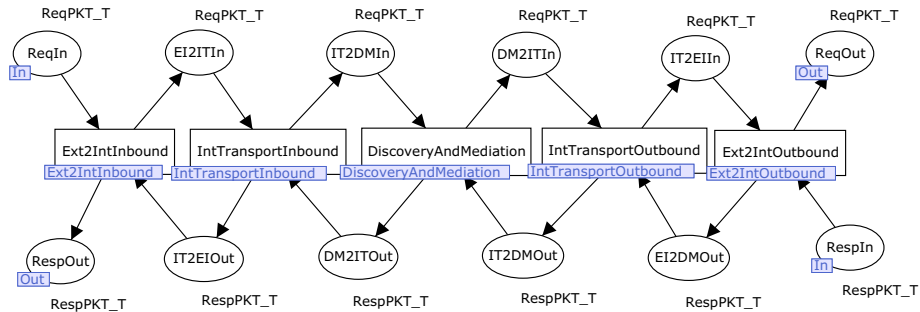


Fig. 4. The components of transport, discovery and mediation mechanism.

These different characteristics of what constitutes a request can be modeled very easily and explicitly by defining appropriate types for the associated tokens. The language for various type (or color in CPN parlance) and function declarations is CPN ML which is based on the functional programming language ML[11]. For our current purposes, a request is treated as a 3-tuple consisting of a consumer request identifier (*CID*), a connection type (*ConnType*) and type of service (*SERVICE*). The declaration of a request type (*ReqPKT*) in CPN ML is specified as:

```

colset ReqPKT =
    product CID * ConnType * SERVICE;

```

CPN supports a timed version by associating a *time-stamp* with each token. The general mechanism for this is to create *timed* color sets. For example, timed request tokens of color set, say, *ReqPKT_T*, can be defined as follows:

```

colset ReqPKT_T =
    ReqPKT timed;

```

We skip the explanation and details of the various other color sets and functions declarations for our CPN model because of space limitations.

The next component is *IntTransportInbound*. This component is responsible for essentially buffering and forwarding the request to the *DiscoveryAndMediation* component, which is the heart and the brain of this architecture. For our present purposes we focus on very simple discovery and mediation mechanism. This will get refined in the subsequent versions of the system, and this is one of the place where we hope the model to guide the implementation. Once a request has gone through service discovery and mediation, it is forwarded to outbound internal transport *IntTransportOutbound* and from there to the outbound external to internal interface component *Ext2IntOutbound*. Response packets simply follow the reverse route, as illustrated.

Using the hierarchical features of CPN, details of these individual components have been created on the associated pages. Next we present details of two of the components, namely, *Ext2IntInbound* and *DiscoveryAndMediation*.

For our present purposes, we are only focusing of one connection type, namely, http. In general though, the connection type on the service consumer side can be different from that on the service provider side and all this could be different from the internal connection type. The internal details of *Ext2IntInbound*, shown in **Fig. 5**, are as follows. It receives the service request (as a CPN timed token of type *ReqPKT_T*) from the consumer. It can then accept this request by firing the *AcceptConnection* transition. CPN provides facility to associate code segments with firing of transitions.

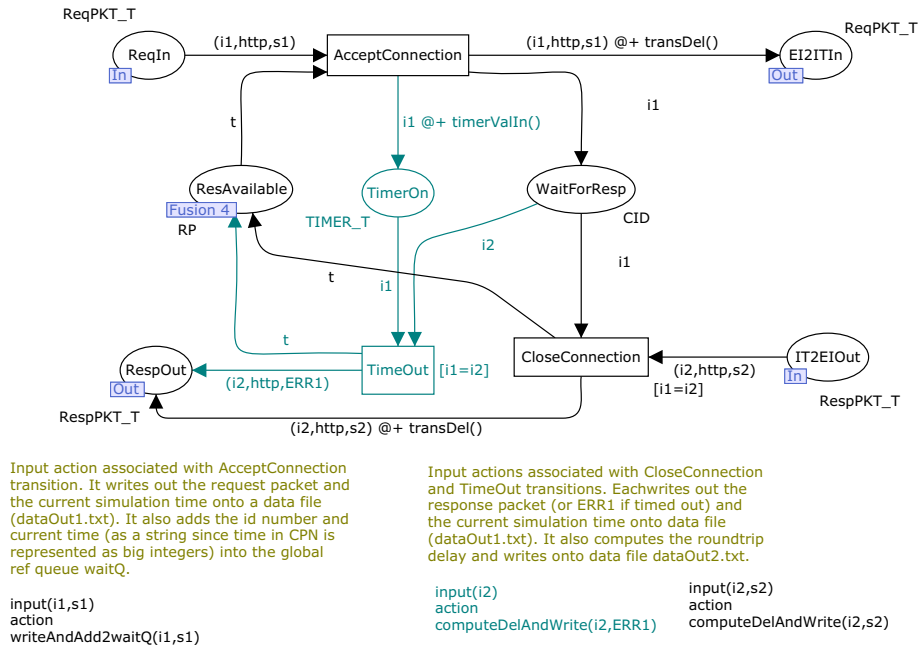


Fig. 5. Detailed net showing activities associated with Ext2IntInbound component.

Here the input action associated with the *AcceptConnection* transition is to write out the request packet and the current simulation time into a data file that can be examined later for desired properties and behavior. If there were any external to internal connection type translations involved, there would be added transitions to take care of those details here. The request is then passed to the internal transport by placing the token in the place named *E12ITIn*. At the same time a token representing the request identifier is added to the place named *WaitForResp*.

In a real-life scenario, each connection that is open consumes some resources. Tokens in the place named *ResAvailable* represent the current number of resources available. A token from this place is removed for each firing of the *AcceptConnection* transition. This represents allocation of a resource (for example a thread from a thread pool). The connection is required to time-out if no response arrives within some specified time-out value. Thus, simultaneously a timed token is put in the place called *TimerOn*. The time stamp of this token represents the time-out value and can be set from a file by making use of input/output facilities of the CPN Tool.

The semantics of timestamp in CPN are that the associated token remains unavailable until the current simulation time becomes equal to or exceeds the timestamp value. Thus, if the response comes in, that is, a token with the correct id value and time-stamp smaller than that of the associated timer arrives, the transition *CloseConnection* fires and the response is forwarded to the consumer by placing the request in the place named *RespOut*. Otherwise, the transition *TimeOut* fires signaling expiration of the timer, and an error response is forwarded. Note that both *CloseConnection* and *TimeOut* have a guard $[i1=i2]$. This ensures that the response or the time-out is

matched with the correct request id. Finally, when either a time-out occurs or a connection is closed, the allocated resource is returned to the pool of resources. This is achieved by adding a token back into the place *ResAvailable*.

Details of the discovery and mediation are given by the net shown in **Fig. 6**. We are currently focusing on a very simple discovery and mediation. In particular, we do not account for mobility of service providers. The function *validReq* : *SERVICE* → *BOOL* is hard coded in that requests for certain services are considered unavailable. In particular, service *d* is considered invalid since in the current test set there are no providers for service *d*. When a request arrives in *IT2DMin*, it is checked for validity and forwarded to next component. This is achieved via firing the transition *forwardRequest* which deposits the request in *DM2ITIn*.

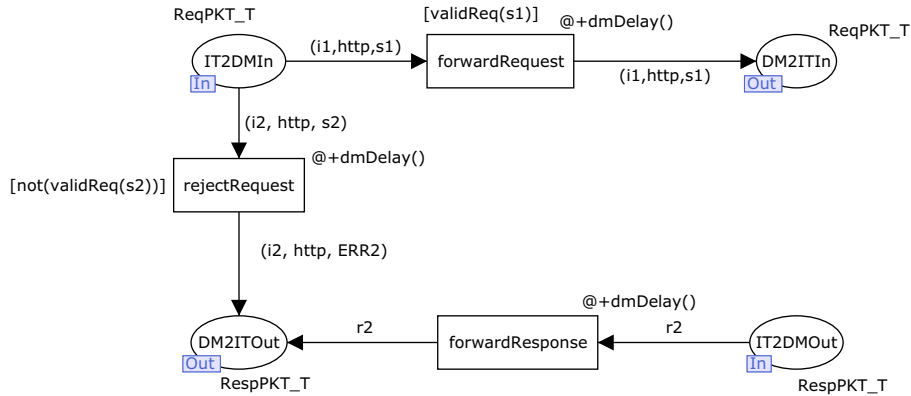


Fig. 6. Net representing details of discovery and mediation component

Note that the facility in CPN to associate data values with tokens and manipulate them or examine them and control actions based on them is a powerful one. Without such a facility it would be difficult to model requirements such as validity of requests, etc. If the incoming request is not valid, an error response is returned and this is indicated by firing of transition *rejectRequest*, which deposits an error packet in *DM2ITOut*. This component is also responsible for forwarding a response packet which is indicated by firing of transition *forwardResponse*. We skip the details of rest of the components here and discuss our verification and validation process next.

6 Quality Assurance, Verification & Validation, and CPN

From industry acceptance point of view and to have the validity of any model predictions incorporated into development, we needed to put in place a well defined verification and validation process for quality assurance. Furthermore, this verification and validation activity was to be carried out not by the modeler but by a third party, which

usually is a Quality Assurance (QA) team associated with a traditional software development process. CPN offers the following four possible analysis approaches:

- Interactive and automatic simulation
- Performance analysis
- State space and invariant analysis
- Temporal logic based analysis of state spaces

Given that a QA person may not be conversant with state space based analysis and given that usually a combination of strategies is required for a meaningful analysis, we decided to start with simulation and performance analysis. Furthermore, CPN provides a full spectrum of input/output facilities and user-defined functions so that we were able to parameterize all relevant input data and read it from data files. This also made it possible for a third party to run simulation and gather data with different input values. CPN Tool also provides an extensive collection of monitoring, performance analysis, and data logging facilities that further simplifies this task [12]. The verification and validation process and approach to integrating modeling into development that we have currently adopted is described in [13]. A simplified view relating modeling, integration, verification, and validation is given in **Fig. 7** below.

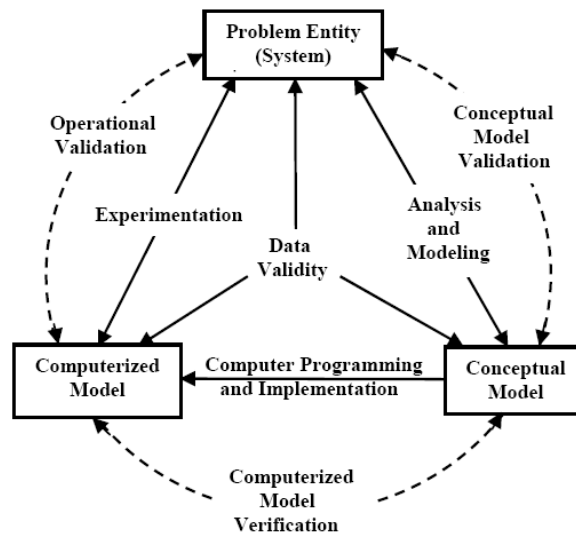


Fig. 7. Simplified modeling, integration, verification and validation process [13].

In our case, the edge labeled *Computer Programming and Implementation* gets realized as a CPN model. Furthermore, what this picture does not communicate is the incremental or spiral nature of the process. Essentially, we repeat the depicted process in each spiral. Our starting point was a base implementation. We then created a model for it. The model was subjected to verification and validation process. The preliminary results from this exercise are presented in the next section.

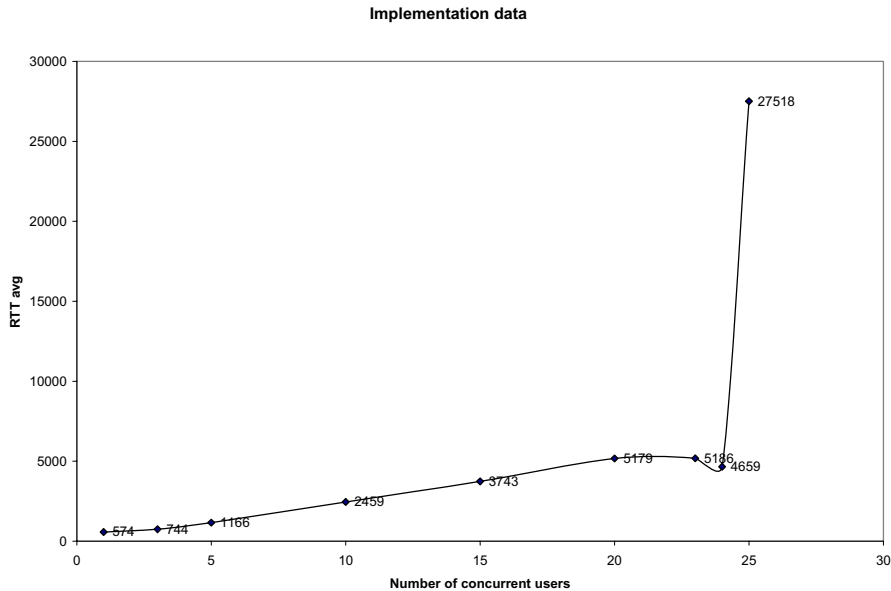


Fig. 8. Average RTT data from system implementation

7 Preliminary Results

Following the approach outlined above we carried out the verification and validation of the created model for quality assessment. To ascertain the validity of the model, we compared the behavior of the model with the corresponding behavior in the real system. For our initial validation attempt, the behavior we chose to ascertain was the performance of the system as the number of concurrent/simultaneous requests was increased. We developed an experiment in the run-time lab to measure the average round-trip time (RTT) of request-response interactions as we varied the number of simultaneous users presenting requests to the system. Performance was thus quantified as average round-trip time.

The experiment was run both on the model and on the real system. Experimental data was used to generate two graphs – one for the model and one for the real system. The shapes of the curves on these two graphs were compared to determine whether the behaviors were similar or not.

The following figures represent the data we collected from our experiments where the underlying resource pool contained 25 possible thread resources. **Fig. 8** shows the performance of the real system, while **Fig. 9** represents data from the model.

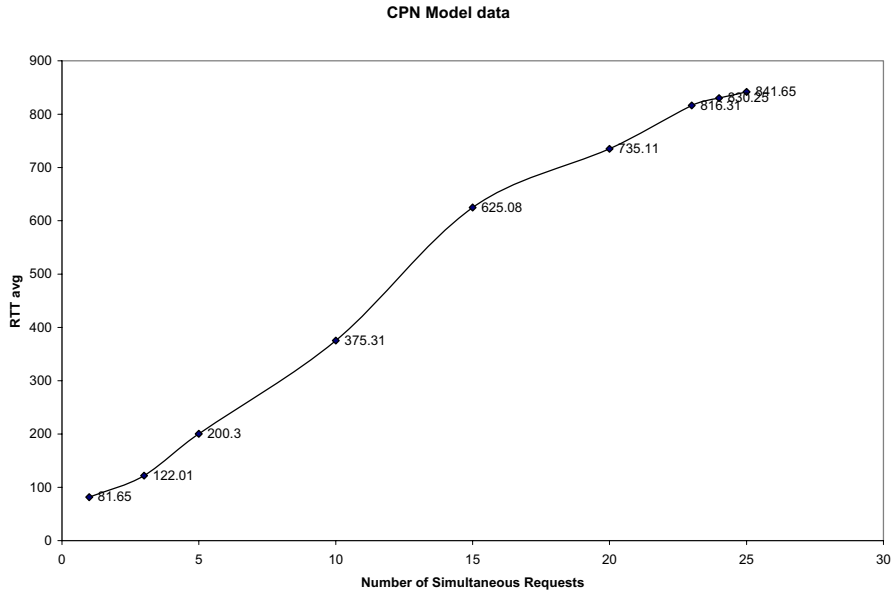


Fig. 9. Average RTT data from model simulation.

We note the following behaviors that are visible in both the system implementation and the model simulation:

- Performance decreases with increasing numbers of simultaneous requests.
- Performance bottleneck occurs when the size of the resource pool available to service requests equals the number of simultaneous requests.

However, a sharp discrepancy exists in the two behaviors when the number of concurrent request reaches the maximum resource pool size. Furthermore, the system implementation could not handle any more requests after this point was reached. The system implementation shows a sudden spike whereas the model data shows a gradual increase. This discrepancy was puzzling to us and our investigation using the CPN model found a bug in system implementation whereby threads for de-queuing operation were being allocated from the same pool as servlet pool creating a deadlock situation. This deadlock situation in the system implementation was later rectified and the results from re-verification and re-validation are given in **Fig. 10** below. It is easily seen that the two graphs show similar behavior. Ideally, these two graphs should coincide. In order for this to happen, the model needs realistic values for each of the parameters it has. However, we are currently limited in terms of what parameter values we can measure in the run-time lab on the real system. We are currently investigating approaches to such value measurements and estimation if real values cannot be measured for all model parameters.

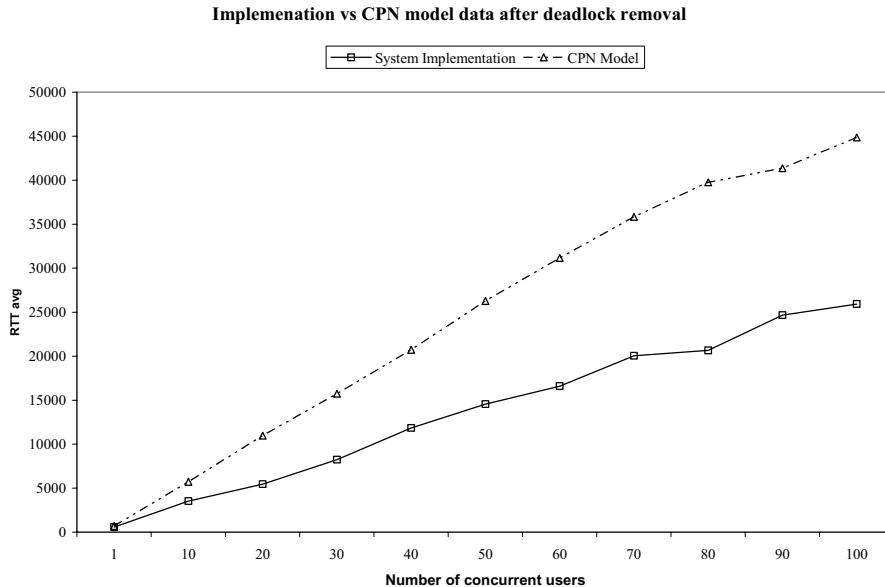


Fig. 10. Avg. RTT data from implementation and model after system deadlock was removed.

8 Conclusions and Future Work

The service oriented architecture (SOA) concept offers a framework for integration of systems and interoperability. This approach is very attractive in many business settings and is especially attractive in a defense setting where the traditional “stovepipe” approach has resulted in poor integration of systems and rendered them non-interoperable. The US DoD has an initiative called Net-Centric Enterprise Solutions for Interoperability (NESI) with the purpose to provide a service-oriented architecture solution approach for defense applications. Thus, many defense operations, including safety-critical ones, are soon to be deployed on a service oriented basis.

A model driven development based approach offers possibility of quality assessment, verification and validation, and quality prediction of such deployments. We presented a service oriented architecture and its model using Colored Petri Nets (CPNs). We illustrated aspects of CPN and the associated modeling and analysis tool called CPN Tool that have made it possible for us to integrate this approach as part of a large-scale defense software development. We also have access to a reference implementation that was used in our verification and validation process. Our preliminary analysis and results revealed a deadlock situation in the system implementation showing an early benefit of model integration in system development. Our future work includes extending the model to include other features and components of the architecture including presence and discovery mechanisms, mobility, and load balancing. Through our modeling effort we hope to guide the current system implemen-

tation and show benefits of a model driven approach in quality assessment, assurance, and prediction.

Acknowledgements: We would like to thank the Gestalt ARCES Team for their help and support. This research was supported in part by the Air Force Materiel Command (AFMC), Electronic Systems Group (ESC) under contract number FA8726-05-C-0008 . The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of USAF, AFMC, ESC, or the U.S. Government.

References

- [1] S. Anand, S. Padmanabhuni, and J. Ganesh, Perspectives on Service Oriented Architecture (tutorial). Proceedings of the 2005 IEEE International Conference on Services Computing, 2005.
- [2] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, and Rawn Shah, Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap. IBM Press, 2005.
- [3] W. Tsai, Y. Chen, and R. Paul, Specification-Based Verification and Validation of Web Services and Service-Oriented Operating Systems. Proceedings of the 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 05), Sedona, 2005, pp. 139-147.
- [4] Net-Centric Implementation, Part 1: Overview (Version 1.1, June 3, 2005), Netcentric Enterprise Solutions for Interoperability (NESI) project, <http://nesipublic.spawar.navy.mil/docs/part1>, accessed Feb. 24, 2006.
- [5] Capability Development Document for Net-Centric Enterprise Services, Draft Version 0.7.15.2, April 9, 2004.
- [6] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Volume 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag, 2nd corrected printing 1997.
- [7] C. Girault and R. Valk, *Petri Nets for Systems Engineering*, Springer-Verlag, 2003.
- [8] W. Reisig, *Petri Nets*. EATCS Monographs in Theoretical Computer Science, Vol. 4, Springer-Verlag, 1985.
- [9] K. Jensen, *An Introduction to the Theoretical Aspects of Coloured Petri Nets*. In: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.): A Decade of Concurrency, Lecture Notes in Computer Science vol. 803, Springer-Verlag 1994, pp. 230-272.
- [10] A. V. Ratzner, et al., CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In W.v.d. Aalst and E. Best (eds.): Application and Theory of Petri Nets 2003. Proceedings of the 24th International Conference on the Application and Theory of Petri Nets (ICATPN 2003). Lecture Notes in Computer Science, vol. 2679, Springer-Verlag, 2003, pp. 450-462.
- [11] J. D. Ullman, *Elements of ML Programming*, Prentice-Hall, 1998.
- [12] B. Lindstrøm and L. Wells. Towards a monitoring framework for discrete event system simulations. In *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES'02)*, 2002. (Also see <http://wiki.daimi.au.dk/cpntools-help/cpntools-help.wiki> .)
- [13] R. D. Sargent, Verification and validation of simulation models, *Proceedings of the 2003 Winter Simulation Conference*, S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds., 2003.