A Case Study in Defining Colored Petri Nets Based Model Driven Development of Enterprise Service Oriented Architectures

Vijay Gehlot and Gayatri Pujari
Center of Excellence in Enterprise Technology
Department of Computing Sciences
Villanova University, Villanova, PA 19085, USA
{vijay.gehlot, gayatri.pujari}@villanova.edu

Abstract

Many businesses as well as government enterprises are moving towards Service Oriented Architectures for their operation. The level of complexity, interaction and interdependence among various components make it difficult to design, develop, and analyze such systems as a whole. In particular, it is difficult to foretell the effect of proposed changes and to evaluate alternative architectures. Adoption of a model-driven approach, coupled with sound Verification and Validation (V&V) techniques can provide a key solution for making qualitative and quantitative predictions about the possible system behaviors.

This paper details a case study in Verification and Validation of a Service Oriented Architecture called MCSOA (Multi Channel Service Oriented Architecture). It is part of a larger exercise to integrate a model-driven approach into MCSOA Software Development for the US Department of Defense. 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.

1. Introduction

With the emergence of Enterprise Service Oriented Architecture (Enterprise SOA) and its adoption by both the commercial sector as well as the US DoD, there is a need to establish a new approach to systems development. Systems built under this approach are not traditional stand-alone systems but a formation of systems of systems that exhibit levels of complexity that are larger than the sum of the individual components. The advantage of such systems of systems is that they can deliver services that are larger than the sum of individual components. In particular, the promise of SOA is a world of cooperating services where application components can be assembled, with little or no effort, into a network of services that are loosely coupled and provide flexible dynamic business processes and agile, adaptable applications. In the DoD context, the significant costs involved in fulfilling the strict and unique defense requirements suggest a need for a model-driven development and quality assurance approach that can accommodate the high degree of concurrency and both synchronous and asynchronous communications in enterprise uses of SOA. However, the challenge is finding a mechanism to understand the effects of local changes on the overall system of systems as well as managing system evolution.

The Object Management Group (OMG) defines Model Driven Architecture (MDA) as follows [1]:

The MDA starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. MDA provides an approach for, and enables tools to be provided for:

• specifying a system independently of the platform that supports it,
• specifying platforms,
• choosing a particular platform for the system, and
• transforming the system specification into one for a particular platform.

The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns.

Model Driven Development (MDD) based on a suitable MDA holds promise not only of easier system development, integration, and maintenance,
but also of the ability to build tools to (semi-) automate construction. An analogous process uses automatic tools for compiler generation starting from language specification. At the heart of model driven development is a suitable modeling language. For a rigorous approach, especially in the context of Enterprise SOA, such a modeling language should be free from ambiguity and be based on the solid theoretical foundation of concurrent and distributed computation. Although there are some Unified Modeling Language (UML) based MDA approaches, they lack direct linguistic support for concurrent and distributed computation with synchronous and asynchronous messaging. Some researchers have pointed out ambiguities in UML. For example, see [2], [3], and [4]. Most UML diagrams are static and hence inadequate for capturing and analyzing dynamic behaviors of SOA systems. Furthermore, UML state machines lack well-defined execution semantics and do not support modeling of multiple instances of classes and, therefore, do not scale well to large systems [5].

To address these deficiencies of UML, we proposed, as part of a contract with the US Air Force, a Colored Petri Nets (CPN) based MDA approach for the Multi-Channel Service Oriented Architecture (MCSOA) that was under development by Gestalt, LLC, within the Distributed Mission Interoperability Toolkit (DMIT) Program managed by the Electronic Systems Center (ESC). This program was geared towards the Net-Centric Enterprise Solutions for Interoperability (NESI) initiative of the US Department of Defense (DoD) and Defense Information Systems Agency (DISA) [6]. As mentioned above, the level of complexity, interaction and interdependence among various components of an Enterprise SOA such as MCSOA make it difficult to design, develop, and analyze systems as a whole. In particular it is difficult to foretell the effect of proposed changes and to evaluate alternative architectures. Thus, key to our adoption of a model-driven approach was integration of sound Verification and Validation (V&V) techniques that can provide information necessary for making qualitative and quantitative predictions about the possible system behaviors.

This paper reports on the V&V aspects or our model-driven development approach as applied to MCSOA development for the US DoD. The goals of the project are to create formalized models of MCSOA node to analyze existing MCSOA behaviors and performance and also to predict the performance and behaviors of planned, future MCSOA architectures. These models are also intended to help assess other SOA implementations and benchmark their characteristics.

The rest of this paper is organized as follows. We briefly describe CPN based MDA Section 2. Sections 3 and 4 detail models and V&V approaches adopted. Details of MCSOA are contained in Section 5 and its CPN model is given in Section 6. Sections 7 and 8 contain model verification and validation results and we conclude in Section 9.

2. Why Colored Petri Nets Based MDA

We proposed an MDA based on Colored Petri Nets (CPNs) because it is a well-studied and fully-formalized visual modeling language having clean semantics with firm mathematical foundation. See [7], [8], and [9] for details. It has direct support for distributed and concurrent processes with both synchronous and asynchronous communication as well as resource constraints and mutual exclusion. In general, therefore, CPNs are a natural choice for creating reference architecture for SOA systems, which are distributed and concurrent by nature. The small set of generic primitives of CPN built around basic concepts eases learning and maximizes applicability. Furthermore, the graphical and visual nature of CPN models eases the learning burden for non-experts. In addition, people familiar with aspects of UML can pick up CPN details fairly quickly.

One attractive aspect of CPN is that it supports both timed and untimed activities and models in a single unified framework. Untimed versions are helpful in creating a first “rough-cut” of the architecture/model when some temporal behavior and performance characteristics are not known. Untimed versions are also useful when the focus is only the functional aspects of the system. In fact, the untimed version is a better approach since one is not forced to add arbitrary time constrains to develop an architecture/model. A key property is that the timed models remain consistent in terms of their expected behavior with their untimed versions under the defined semantics of CPN.

CPN models are built using a software tool called CPN Tools, which is available under a free license and with support from an active development team. This reduces both initial and maintenance cost of CPN based development. CPN Tools supports both simulation-based as well as state-space based analysis in a single unified framework. This reduces the cost of having to acquire or build separate tools for separate analysis purposes. As an added feature, CPN Tools has a visualization framework built on
top that can be used for domain specific visualizations.

3. Model, Verification, and Validation

A model is a representation or abstraction of a physical system or process and it is built to understand the functionality, structure and the behavior of the system. If a model does not flawlessly depict the actual system, any conclusions derived from the model are likely to be invalid. Hence it is necessary to determine that a model accurately represents the conceptual description and design specification. Model verification and validation (V&V) are therefore essential parts of the model development as well as model-based development process [10] [11].

US DoD defines Verification and Validation activities as follows [12]:

**Verification** - The process of determining that a model or simulation implementation accurately represents the developer's conceptual description and specification. Verification also evaluates the extent to which the model or simulation has been developed using sound and established software engineering techniques.

**Validation** - The process of determining the degree to which a model or simulation is an accurate representation of the real-world from the perspective of the intended uses of the model or simulation.

4. Adopted V&V Approach

Our approach to verify and validate the models was adopted from [13] and is detailed below.

4.1 Verification Approach

The two approaches followed for the verification of the models are as described below:

1. **Specification verification** - is assuring that the software design and specification for programming and implementation of conceptual model are satisfactory. This included verification of the requirements by the MCSOA experts.

2. **Implementation verification** - is assuring that the simulation model has been implemented according to the Simulation Model Specification. The recommended techniques consist of animation, structured walk-through, tracing techniques. When possible, develop automated tests to ensure reproducibility, and objectiveness. We conducted three types of verification experiments. Here is a brief overview of these experiments.

A. **Tracing**: The Modeler conducted this experiment by following the behavior of specific entities through the model, to verify if they function according to the requirements specified.

B. **Test Procedure Execution**: Model verifiers carried out these experiments. Test procedures were created for the model and model verifiers executed those against the model.

C. **Requirement Traceability Matrix**: QA lead and the model verifiers completed the Requirement Traceability Matrix document, by tracking the models back to the requirements to show accuracy of implementation.

The details of the Test Procedure execution and the Requirement Traceability Matrix document completion are described in section 7.

4.2 Validation Approach

Our approach to validation was to compare the behavior of the model with the corresponding behavior in the real system. If the behaviors of the model and the real system were similar then the model would be considered valid.

For the validation of the model, the ARCES project followed a “Real world paradigm” approach, since the real-world system (MCSOA) exists and its understanding is desired. The following are types of model validation that were used for the validation of MCSOA node model.

1. **Face Validation**: This involved gathering of opinions about the reasonableness and accuracy of the model by having meetings with MCSOA system experts.

2. **Parameters Variability/Sensitivity Analysis**: This was an iterative process and it required model tuning. Data was collected by changing the values of the input and internal parameters of the model. The behavior of the model and the output data was analyzed. These experiments were also performed on the real system to check if the same relationships occurred in the model.

3. **Predictive/Event Validation**: Model Validator collected data from the MCSOA test bed to determine if the system’s behavior and the model predictions are the same. The events occurrences
in the model were compared with the distribution of these in the system for typical system events. The details of the validation strategies and the test results are described in section 8.

5. The Multi-Channel Service Oriented Architecture (MCSOA)

MCSOA is an enterprise service bus (ESB) that acts as middleware that brokers and routes requests from consumers to providers and forwards responses from providers to consumers. It is a development, deployment and discovery framework that can be used to build, deploy and run SOA applications. The internal architecture of MCSOA node is depicted in Figure 1.

It provides three channels for the communication between the consumer, broker and provider. Http, SIP and JMS are the three communication channels available for the inbound traffic (messages between the Consumer and Broker) and outbound traffic (messages between the Broker and Provider). The exchange of messages can take place either over homogeneous or heterogeneous inbound and outbound channels. For example an Http Consumer can request a service provided by an Http Provider (homogeneous). The Http Consumer can also request a service provided by a JMS Provider or SIP Provider (heterogeneous).

When a Consumer requests for a service, the Broker finds the Provider, which is capable of satisfying that request. Then this request is routed to the specific Provider by the routing engine and the response obtained from the Provider is sent back to the Consumer, via the Broker.

Although MCSOA is a multi-channel architecture, the study here concentrates on the Http channel because of its widespread use. The base MCSOA model was created by focusing on the single communication channel, the Http channel for MCSOA usage. Figure 1 shows the Base MCSOA node.

6. CPN Model of MCSOA Node

In this project, the model-driven approach was to be integrated into an existing software development of MCSOA. Thus, the first “reverse-engineering” task was to build model artifacts of existing MCSOA components. Thus, after referring to the MCSOA documentation and discussions with the developer communities of MCSOA, the requirements for basic MCSOA node were put together which became the basis for MCSOA model development.

Each MCSOA node is composed of several components, which provide the functionality of the MCSOA node. Behavior of each of these components was analyzed in order to develop the models that will represent the real system. The various sub components comprising the base MCSOA node are depicted in Figure 2.
Consumer and the Http/SOAP Provider. The Inbound Http Channel receives requests from the Consumer. It adjusts the request format to the internal MCSOA message format (SIP). It forwards the message to the next entity in the processing path (until it reaches the Service Broker), waits for the response to come back, transforms the response from the internal MCSOA message format to Http format and handles the response to the Consumer.

6.1.2. The Broker Queue. The Broker Queue is a FIFO (First in First out) type of queue which facilitates a point-to-point (PTP) exchange of messages between the Service Broker and Inbound and Outbound Channel entities. In this type of message exchange each message is addressed to a specific queue, and receiving clients extract messages from the queue(s) established to hold their messages. Queues retain all messages sent to them until the messages are consumed or until the messages expire.

6.1.3. The Service Broker. The Service Broker component of the MCSOA node is the intermediary in the Consumer-Provider message exchange path. Its role is to route requests and responses.

6.1.4. The Http Controller Queue. Http Controller Queue component of the MCSOA node participates in the routing of requests that have been matched to a capable provider. It is the means by which such requests are passed from the Service Broker to the Outbound Http Channel model.

6.1.5. The Outbound Http Channel. The Outbound Http Channel component of the MCSOA node is an entity responsible for facilitating communication between the Http/SOAP Provider and the Service Broker capability within the MCSOA node. The Outbound Http Channel receives the request message from the Service Broker, transforms the internal MCSOA format (SIP) to the Httip format and sends the request to the Provider. The Outbound Http Channel acts as an Http client from the point of view of Provider. The Http Outbound Channel waits for the Provider to satisfy the request and receives its response in Http format. It translates the Http format to internal MCSOA format (SIP) and forwards the response to Broker Queue.

6.1.6. The Service Topic. The Service Topic component of the MCSOA node participates in the Consumer-Provider exchange of messages. It facilitates publish/subscribe (pub/sub) exchange of these messages between the Service Broker component and other actors in the message routing path. In this pub/sub type of message exchange, clients publish messages to a topic and clients subscribe to messages that they are interested in receiving. Publishers and Subscribers are generally anonymous and may dynamically publish or subscribe to content made available on the topic. The system takes care of distributing the messages arriving from a topic's multiple Publishers to its multiple Subscribers. Topics retain messages only as long as it takes to distribute them to current Subscribers.

The complete sets of requirements are contained in [14].

6.2. Design of MCSOA node Model

This section gives a high-level overview of the design of the MCSOA node. The architecture is based on the requirements as described above. To create models and to analyze MCSOA behavior and its performance, CPN Tools software suite was used. We used CPN’s hierarchical module support as well as enhancement to token color sets to implement model refinements. The presented model captures the behavior of HTTP/SOAP request exchange between Service Consumer and managed Service Provider communicating over single MCSOA node.

Figure 3 shows the top-level abstract view of the model. There are three primary system entities: Consumer, MCSOA Server and Provider. Each of these three entities represents an additional level of the model containing further details of the model.

![Figure 3: Top-level model view of MCSOA](image-url)

Network communications are represented through the model in the form of packets, in this case ReqIn and ReqOut are incoming and outbound...
request packets, and RespIn and RespOut are incoming and outbound response packets.

The high-level model of the MCSOA Server is depicted in Figure 4, showing the relationships among, and communication paths between, the various lower-level components of the system model.

The high-level model of the MCSOA Server is shown in Figure 4. The incoming request packets are handled by the MessageBusIn, and the outgoing response packets are handled by the MessageBusOut. Communication paths between these components are indicated by the channels labeled ChannelIn and ChannelOut. The broker component is responsible for translating the incoming HTTP requests into SIP responses and sending them to the MessageBusOut. This translation overhead is handled by the MB2BIn and MB2COut transitions. The current simulation time is written to a data file (dataOut1.txt). It also adds the CID number and current time (as strings since time in CPN is represented as big integers) into the global ref. queue wait Q. The transition translates the http channel into sip and it also adds a delay for this translation. The request is then passed to the Channel2MessageBusIn (C2MBIn), which is handled by the MessageBusIn.

The response comes in from MessageBus2ChannelOut (MB2COut) and fires the CloseConnection transition. The CID of the response is compared to the CID in the WaitForResponse state and if successful, the connection is closed and the resource is put back in the resource pool. The response is then translated from sip to http and sent to ResponseOut (RespOut), which will be handled by the Consumer.

The design details of MCSOA model artifact are described in [15].

7. MCSOA Model Verification

Verification of the MCSOA HTTP exchange CPN model (MCSOANode-1.2Timed.cpn) was carried out executing a set of test procedures and completing the requirements traceability matrix.

7.1. Test Procedure Document

A test procedure document was created for every model component, using a test procedure template. A test procedure document was named to reflect the component name and ID from the software requirements document and requirements traceability matrix. Each test procedure was executed by a model verifier, a team member other than the author of the model being verified.

7.2. Requirement Traceability Matrix

The requirements traceability matrix is a table used to trace and verify that all stated and derived requirements are allocated to system components. It is the document which maps requirements in the requirements document with test cases. Traceability Matrix is a proof of document to ensure that all the specifications have been tested and the model being designed is error free.

The requirements traceability matrix contains the following fields:

1. A unique identification number assigned in ascending order.
2. Cross-reference to the section in the Requirements Document that contains the detailed description the requirement.
3. High level description of the requirement.
4. The name and version of the model created that covers the requirement.
5. The test procedure(s) created to verify the model.

For example, given below is a part of the Requirements Traceability Matrix used for the verification of MCSOA http exchange model.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>ARCES</th>
<th>Description</th>
<th>Model</th>
<th>Associated Test Procedure(s) for Verifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inbound Http Channel model shall provide the ability get the http request from the Consumer.</td>
<td>ARCES</td>
<td>1.2Timed.cpn</td>
<td>ARCES_Inbound_HTTP_channel_3.2.1_jp, step 1</td>
<td></td>
</tr>
<tr>
<td>Inbound Http Channel model shall allocate a resource from the resources pool for each Consumer request.</td>
<td>ARCES</td>
<td>1.2Timed.cpn</td>
<td>ARCES_Inbound_HTTP_channel_3.2.1_jp, step 2</td>
<td></td>
</tr>
</tbody>
</table>

7.3. Steps followed to carry out the verification task

1. Start the CPN tool and load the XML file containing the model. In the model from (To load the model, drag the Net toolbox onto the design screen, then click on the Load Net icon that invokes a standard Windows file open dialog.) Upon loading, initialization of the model is performed using definitions generated during the design of the model.

2. Drag the simulation toolbox on the model and select the tool which executes a transition, by clicking on it.

3. Make sure that all the subnets containing the components of the model are opened.

4. Follow the steps in the test procedure document for each of the model components and for every requirement stated in the software requirements document, check the result of the action performed. If it is the expected result as stated in the test procedure, put ‘passed’ in P/F column in the test procedure and the appropriate number of the related requirement in the Related requirement (from traceability matrix) column.

5. Follow above step for each component of the model and hence execute all the test procedures. As an example, given below is a fragment of the test procedure used to verify the MCSOA http exchange single node model.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Expected Result</th>
<th>P/F</th>
<th>Related Requirement (from Traceability Matrix)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fire 'AcceptConnection' transition on Channel In page.</td>
<td>The token should be removed from ReqIn, and deposited in C2MB In.</td>
<td>Passed</td>
<td>3.2.1.1</td>
<td></td>
</tr>
<tr>
<td>Fire 'AcceptConnection' transition on Channel In page.</td>
<td>RP (resources pool) should decrease.</td>
<td>Passed</td>
<td>3.2.1.2</td>
<td></td>
</tr>
</tbody>
</table>

The model was verified against the requirements. The detailed test results of these verification experiments are contained in [16].

8. Model Validation and Results

For the examination of the inbound HTTP channel in the MCSOA node model, we developed an experiment to measure the average round-trip time (RTT) of request-response interactions by varying the number of simultaneous users sending requests for services, to the system. As the number of simultaneous users was increased, the behavior of the system was analyzed. This experiment was run on the model and the real system and graphs were plotted using the output data. The shapes of the curves on both the graphs were compared to determine whether the behaviors were similar or not.

To facilitate data collection, the model utilized the I/O capabilities of CPN Tools. Input files were used to set various parameters for the model. Information was written out to output files during model execution for data analysis.

In these validation experiments, the model was configured to create a timeout of an inbound request on the inbound HTTP channel following the completion of 1,000 simulation time units (stu). The model was also executed for all 9 inbound load values (i.e. number of simultaneous users as 1, 3, 5, 10, 15, 20, 23, 24, and 25) with the timeout set for 10,000 simulation time units.

To obtain the performance data needed for validation, an instance of the system being modeled was deployed in the runtime lab. Each of the logical components was deployed to its own machine. Both the MCSOA node and the provider are Java J2EE applications and were deployed to WebLogic instances on Dell PowerEdge 2850 and Dell OptiPlex GX280 machines respectively. The consumer was deployed on Dell Latitude D600 machine. The provider was the HelloWorld sample web service provided with BEA WebLogic 8.1 domain running on the GX280. Parasoft’s SOATest software version 4.1 was used as the consumer component. SOATest has the load-testing feature, which was used to simulate the behavior of multiple, simultaneous users submitting requests for service from a provider via
the MCSOA node. It allowed up to 100 virtual users to be simulated.

The primary experiment to measure the average round-trip time (RTT) of request-response interactions by varying the number of simultaneous users sending requests for services, to the system was performed using this configuration.

Our initial validation experiment data showed some surprises. In particular we discovered a software deadlock situation via disparity of model and software behavior in the collected data. The model was useful in rectifying this unsafe behavior. The development team updated the software and we carried out the validation experiments again. A sample plot showing consistency of MCSOA model and MCSOA software is given in Figure 6 below.

![Figure 6: MCSOA model and software behavior](image)

9. Conclusions

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 our experience with Colored Petri Nets based model driven development of an ESB, called MCSOA, for defense applications.

Through this work we have shown suitability of CPNs as a modeling language for model-driven development of SOA systems that are concurrent and distributed in nature and where both synchronous and asynchronous communications as well as resource constraints are key aspects of system functionality. Through our modeling effort we hope to guide the current system implementation and show benefits of a model driven approach in quality assessment, assurance, and prediction.

Our future work includes developing techniques and tools to support a wide range of model evolution activities, including model comparison, model refactoring, model inconsistency management, model versioning, and co-evolution of models.

Acknowledgements: This work 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.

10. References


