Use of Colored Petri Nets to Model, Analyze, and Evaluate Service Composition and Orchestration

Vijay Gehlot and Kranthima Edupuganti
Center of Excellence in Enterprise Technology
Department of Computing Sciences
Villanova University, Villanova, PA 19085, USA
{vijay.gehlot, kranthima.edupuganti}@villanova.edu

Abstract

The Concurrency, communications, resource constraints, and quality of service attributes are distinguishing features of web services. Assessment and determination of impact of the nonfunctional aspects such as service granularity, governance, composition, and orchestration is an overarching concern and this exercise should be carried out at the architecture design stage rather than post implementation and deployment. In this paper we illustrate, by means of a simple example, the use of Colored Petri Nets (CPNs) to model service composition and orchestration. We use the associated software tool called CPN Tools to perform the analysis. The results can be used in many ways such as to determine design alternative or to check conformance with existing service level agreements, etc. Colored Petri Nets, being a graphical modeling language suitable for modeling distributed, concurrent, deterministic and nondeterministic systems with synchronous and asynchronous communications, offer a natural choice for this endeavor. Although the example is given in the context of web services, CPNs view of interaction and coordination of systems is abstract and can be applied to different notions of services equally well.

1. Introduction

Web services and service oriented architecture (SOA) are the key technologies that underlie Service Computing, which itself is closely related to the emerging important area of Service Sciences, Management and Engineering (SSME). The SOA framework consists of three components: a service provider, service consumer and a directory/discovery mechanism [16,4]. A recent report [2] distinguishes two styles: static and dynamic. In the former case, the service interface and location are assumed known and bound statically when the service consumer is implemented or deployed. The dynamic scenario offers more flexibility since the directory is queried by the service consumer at runtime for the provider’s address and the service interface. This result in run-time binding and after acquiring the required information, the service consumer can invoke the operations of the service provider. Therefore, this paradigm utilizes services as the basic construct to support the development of distributed applications in heterogeneous environments. The advantages of this approach include reusability, inter-operability, scalability, flexibility, and cost efficiency. This paradigm envisions a world of cooperating services where components can be assembled with little or no effort, to meet the changing needs, yielding loosely coupled, dynamic, and agile applications. However, there are many challenges to overcome before this promise becomes a true reality. A recent report [7] on service innovation under its recommendations for research suggests “Create modeling and simulations tools for service systems.” and emphasizes that “Perhaps more than any other subjects, advancement in Service Science depends on models and simulations of alternative service systems designs. When data are not readily available, service practitioners need simulation tools to support their decision-making processes.” Another research has identified “computational theory and modeling of service systems” as some of the Big Services Problem [14] and it too emphasizes that “we need computational theories to generate models that help us understand and guide how services associating humans and information and communication technologies (ICT) emerge and how they help organizational structures emerge, interact, evolve and adapt to better meet the needs and aspirations of people, business, and society.”

It is our view that, due to the very nature of the problem, there will be many possible solutions to each of these challenges and that the use of appropriate modeling and analysis techniques can help in
evaluation of such solutions and give a better insight into proposed solutions at the design stage. In particular, executable models can be used to quantitatively analyze solutions to Service Foundations and Service Composition as described in [17]. Another related recent effort identifies research issues from the perspective of SOA adoption [12]. Their overall research framework divides the issue into three spaces: Problem Space, Planning Space, and Solution Space. Here too modeling can be helpful in exploring the SOA strategy and plans in the Planning Space as well as the Solution Space. Furthermore, [2] gives general guidelines on evaluating a SOA based on current implementation approaches. We believe that our model-based approach can both complement and supplement their general qualitative guidelines with quantitative data.

In this paper we introduce a Colored Petri Nets (CPNs) based approach to modeling and analyzing SOA solutions. Towards this end, we consider a simple scenario. Although hypothetical in nature, our scenario can be seen as an abstraction of all four subcategories in the Service Composition category [17]. The rest of this paper is organized as follows. We summarize some related work in Section 2. Section 3 gives a brief overview of CPNs. Our example scenario is explained in Section 4. We build and describe a CPN model of this example in Section 5. Section 6 contains a discussion of our results and we conclude in Section 7.

2. Related Work

Over the years there have been several research efforts to model and analyze Web services. Two reported works that are closely related to our work presented here are [15,3]. The first addresses the impact of slow services on the overall response time of a transaction that uses several Web services in parallel. The methodology employed is to develop a quantitative model to describe key aspects of the modeled system numerically and represent relationships via equations. However, a limitation of this approach is that a closed form solution for a general composition involving a variety of attributes and behaviors is typically not possible. Such an approach can serve as the first step in the modeling to provide a basic understanding of the modeled process and can be used for intermediate validation.

The second work uses timed automata and an associated tool for describing and analyzing the behavior of Web Services. In particular, they use a scenario consisting of a traveler, a travel agent, and an airline reservation system.

Service execution is determined by an underlined work flow and CPNs have been used for workflow modeling. In particular [1] reports on deriving a Business Process Execution Language (BPEL) implementation from requirements specified using colored work flow nets. A similar exercise is carried out in [11] for an electronic patient record system.

3. Colored Petri Nets

Petri Nets provide a graphical modeling language or notation well suited for concurrent and distributed systems in which communication, synchronization, and resource sharing play an important role [20]. A Petri Net consists of places (denoted as circles or ovals), transitions (denoted as rectangles) and arcs (denoted as arrows) that connect a place to a transition or a transition to a place. Places may have tokens and firing of a transition removes tokens from its input place and adds tokens to its output place. The firing of a transition is an abstraction of occurrence of an event and movement of tokens describes state change. This small basic vocabulary and simple semantics renders Petri Nets very flexible in terms of application domains for modeling. Colored Petri Nets (CPNs) are extensions of Petri Nets that allow modeling of both timed and untimed activities and, from a practical point of view, include a high-level programming language as their inscription notation and support a powerful module mechanism that allows building of models in hierarchical manner. The Petri Net core provides the primitives for process inter-action, while the programming language provides the primitives for the definition of data types and the manipulations of data values. These enhancements yield, in general, compact and structured models when compared with ordinary Petri Nets. Figure 1 and Figure 2 illustrate some useful net configurations. Figure 3 is an illustration of a complex composition built using the simpler net
specifications of Figure 1. This composition syntactic in nature in general, service composition would be semantic in nature for example, the choice situation may depend on computation of some semantic values. Such semantic compositions can be expressed in CPNs using the underlying inscription language called CPN ML.

As with Petri Nets, CPNs have a fully formal, mathematical definition and a well-defined syntax and (dynamic) semantics making CPN models executable. This formalization is the foundation for the different a behavioral properties and the analysis methods. The complete formal definition of CPN 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 graphical software tool such as CPN Tools [18]. This tool allows one to create a visual representation of a CPN model and has facilities for creating, editing, simulating, monitoring, and analyzing models. Both simulation-based as well as state-space based (model-checking) analysis methods are supported by this tool. Given this, we now have a single formalism and an automated tool to explore both functional and performance properties of a system by resorting to untimed and timed versions of the model of the same system. The main difference between an untimed and timed CPN model is that in the latter case the tokens carry a time stamp denoting the earliest (model) time at which the token can become available. With this approach, the possible behaviors of a timed CPN model always form a subset of its untimed behavior. Thus, turning an untimed model into a timed one cannot add new behavior. As a result, if is often convenient to start by building an untimed CPN model to investigate the functional and logical properties and then add time for performance analysis and to investigate quality of service attributes.

The CPN Tools graphical editor allows the user to create and layout the different net components. One of its nice features is the use of binders and pages to visually divide a model into components, enhancing its maintainability and readability without affecting the execution or analysis of the model. For hierarchical modeling, both bottom-up as well as top-down approaches are supported. CPNs and CPN Tools have been used in numerous practical projects in a large variety of application areas [19]. An excellent introduction and practitioner’s guide to CPN and CPN Tools is [10].

4. Dynamic service composition example

To illustrate the use of CPN and CPN Tools in modeling and analyzing Web services we consider a simple case as shown in Figure 4 which has consumers and providers, where the objective is to do a cost benefit analysis of various options available for service composition and orchestration. Although hypothetical in nature, this case can be seen as an abstraction of many real life service compositions. For example, Figure 4 can be interpreted in a health care delivery and maintenance system where we can view patients as consumers, a local health center(Hospital A) as a local service provider and the larger health center with a trauma facility (Hospital B) as a remote service provider. Services for specialized care or in case of a disaster situation or an emergency can be facilitated by Hospital B if such services are not available at Hospital A. These two health centers can themselves be a part of a larger Regional Health Information Organization (RHIO). Another interpretation of Figure 4 is a library system. If the books are not available locally then the request can be placed to another library.
which is the remote provider, which can either accept the request or decline it based on the availability. Note that the focus is the evaluation of Web service architecture from the point of view of integration rather than the internal design of software that implements a service. Let us assume we want to create some Web services and we have a fixed set of computational resources. In addition, we have a choice of making use of a third-party service provider for some of the functionality we want to create. But this option has an additional cost per use. Before coding and deploying our services, we would like to evaluate each of these options in terms of cost, customer satisfaction, etc. It is assumed that for our customer base, we have a network with higher bandwidth and throughput whereas making use of third-party contracted services definitely has a higher delay factor. It is also assumed that the third-party provider has better and more computational resources. A middle ground could be making use of contracted services only when we cannot handle those because of system overload. Thus, we’d like to get an assessment of our capacity and threshold of reaching out to contracted services. In terms of the terminology used in [2], our example scenario falls in the static category since we are not considering an explicit discover step here. In the next section we develop and describe a hierarchical CPN model to evaluate and assess these scenarios.

5. The CPN Model

CPNs support a powerful module mechanism to support hierarchical construction of a model. The hierarchical approach of CPN allows for separation of concerns in a semantically clean manner and, therefore, Consumer, Network, and Provider in Figure 5. The detailed activity associated with each of these entities can then be represented on a subpage associated with it. Such transitions are called substitution transitions. In CPN Tools, a substitution transition is shown with a double border.

The arcs in this figure summarize the overall flow in the system, that is, service request messages flow from response messages flow from the provider to the consumer to the provider via the network and consumer via the network. For our present purpose, a request is a tagged pair consisting of a request identifier and a service identifier. In this paper we focus on service requests and responses at a higher level and hence a service is denoted by just a number and the response is the same as the service number. In CPN, the (type) declaration for tokens is given using a color set. The relevant color set declarations for messages are given in Figure 6. Note that the CPN color set declarations allow for an arbitrary structured type declaration and, therefore, if we were to represent attributes of a service requests and responses, we could employ a more structured color set declaration. For example, the structure of an entire WSDL or a SOAP message can be represented as a color set in CPN.

```
colset SERVICE = int with 1..5;
colset ID = int; colset SID = product ID * SERVICE; colset MSG = union R: SID;
colset MSG_T = MSG timed;
```

Figure 6: Color set to encode message tokens in the model

The activities associated with a consumer are modeled in the subpage Consumer shown in Figure 7. In this net, the place Ready contains a timed token. The time stamp of this token denotes the earliest time the next request will be sent. A service request is sent by firing the transition SendRequest. The generated request (token) consists of a request identifier and a service and added to place Request1 denoting sending of the request into the network. The place NextID keeps track of the next request number. Each time a
When a request is sent, this number gets incremented by one. This is denoted by the increment expression on the associated arc. Once a request is sent, a token consisting of the time of the request and the request itself is added to place Wait denoting the activity of waiting for rely. In this simplified system, we do not model any timeouts. Thus, a request may wait for arbitrary long time for a response. When a response is received from the network, that is, a token appears in the place Response2, it is matched for the id numbers before it is accepted by the firing of the transition ReceiveResponse. This is controlled by the guard \([i_1=i_2]\) associated with this transition. In general, a guard may be any arbitrary sequence of boolean expressions belonging to the underlying programming language CPN ML which itself is based on the popular functional language SML [22]. The high level nature of CPN ML allows, for example, easy coding of complex business rules and service level agreements (SLAs) when modeling such systems with CPN.

We have assumed the network to operate under ideal conditions. That is, we assume no loss or duplication of messages and infinite capacity. If we were to relax these assumptions, we could take advantage of the modular approach supported by CPN and enhance the net work behavior without affecting the other components of the system. The ideal network under the present assumptions is depicted in Figure 8. The inscription \(@+tDel()\) on transitions ForwardReq adds a delay to the time stamp of the message token removed to account for transmission delay.

The final component of our model is the service provider. In our present example we want to be able to create two different scenarios. One where all services will be deployed locally. Another scenario is the service contract agreement with a third party service provider that we may want to make use of if the current provider gets overloaded. Provider module shown in Figure 9 depicts the first scenario. The service provider has provision for different types of services as governed by the associated token type SERVICE. For the purposes of this paper, this type is a sub range of integers. For a more elaborate model, other attributes of the service, including semantic annotations and quality of service annotations can be encoded as a suitable color set describing a service token [21,23]. In the net depicted in Figure 9, a request received from the network (i.e., a token in place Request2) is accepted and serviced only if there are enough resources available. Currently, the only resource of concern is the number of threads represented by the place ThreadsAvail. The guard \([th<>0]\) associated with the transition AcceptRequest and transition firing semantics of the CPNs capture this constraint to the place ProcessRequest. The time stamp of this token represents the processing time associated with the request.

Once a request has been processed, the response/result is sent by adding a token to the place Response1. Simultaneously, a thread in which the service was executing is returned to the pool of available threads. To create the second scenario where
we want to send off the request to a third party service provider for processing when there is an overload, we replaced the provider module shown in Figure 9 with the one given in Figure 10. Here the substitution transition HandleRequest is expanded into a subnet containing a choice of either handling a request locally or remotely. This subpage is shown in Figure 11. The substitution transition HandleLocally expands exactly to the provider module discussed before. The substitution transition HandleRemotely is expanded to the net shown in Figure 12. For sake of brevity we do not elaborate in terms of a subnet describing the remote routing and proxying. The guard \( [\text{th}=0] \) on the transition ensures that the request will be forwarded to a third-party provider when there are not enough resources available locally. The delay inscription \( @+\text{rprDel()} \) factors the remote transmission as well as remote processing delay.

### 6. Results

We made use of CPN monitoring facilities [13] to extract and collect data from the model during simulation runs. Marking size monitors and generic data collector monitors were used for thread counts, pending requests and round trip delay (RTT). The data values that are collected by data collector monitors may not be independent and identically distributed (IID). For this reason, CPN Tools include automatic

![Figure 10: The service provider module for scenario two](image)

![Figure 11: The module for handling request locally or remotely.](image)

![Figure 12: The module for handling request remotely.](image)

![Figure 13: Comparison based on round trip time.](image)
we note that the case with local only provider has a
sharper and steeper curve and there may not be any
advantage of going in for remote service if the inter-
arrival rate is more than 4 units.
Finally, we plotted the maximum delay of
processing a service request under the varying load
conditions. This graph is shown in Figure 15 once
again we see that the maximum delay experience may
suddenly become unacceptable if the inter-arrival rate
is 2 or less.

7. Conclusions

Using a simple example, we have argued and
demonstrated that Colored Petri Nets are a natural
choice when it comes to modeling and analysis of
Service Composition and Orchestration. The
distinguishing characteristics of web software is the
distributed and concurrent nature of user/service
requests, the synchronous as well as asynchronous
nature of coordination and communication, resource
constraints, and overheads. Petri nets were developed
precisely for modeling such systems and scenarios. In
fact, the graphical nature of the language gives a very
visual representation of sequential and parallel compo-
sition, both asynchronous and synchronous
communication, resource constraints, and mutual
exclusion. Colored Petri Nets extend the vocabulary of
ordinary Petri Nets and add features that make them
suitable for modeling large systems. CPNs combine the
strengths of ordinary Petri Nets with the strengths of a
high-level programming language. Petri Nets provide
the primitives for process interaction, while the pro-
gramming language provides the primitives for the
definition of data types and the manipulations of data
values. From a practical applications point of view,
CPNs support a mechanism of modules that allows one
to construct models of large systems in a hierarchical
manner. The hierarchy and module concept of CPNs
allow different levels of abstraction that are inherent in
most systems. In fact we have successfully employed
CPNs to model and analyze both the internal
components and interactions as well as the external
user/service request patterns and behaviors of an
Enterprise Service Bus (ESB) developed defense
applications [5,6]. 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 and
other quality of service attributes of a system. The
associated graphical software tool (CPN Tools)
provides support for construction as well as analyzing
CPN models. We used the simulation-based analysis
supported by CPN Tools to analyze and compare the
two scenarios of our example. For illustration purposes
we had selected known distributions for service
requests and processing. For these distributions results
can be derived analytically. However, if service request
and processing have varying distribution or non
standard patterns then analytical solutions cannot be
found and simulation based environment as discussed
in the paper would be the only possible approach.
Furthermore, added advantage of using CPN for
simulation is that it supports state-space based (model
checking) analysis techniques that are not available in
other simulation tools. In our view, CPNs together with
CPN Tools, provide a unified framework to model both
timed and untimed activities and to perform both
performance and other quality of service analysis as
well as functional analysis of Web software within
single setting.

8. References

go all the way: from requirements via colored workflow nets
to a BPEL implementation of a new bank system. In


