CIT. Journal of Computing and Information Technology, Vol. 24, No. 3, September 2016, 253–269
doi: 10.20532/cit.2016.1002787

Dependency Modeling of a SOA Based System Through Colored Petri Nets

Pawan Kumar and Ratneshwer Gupta

Department of Computer Science (MMV), Banaras Hindu University, India

Dependency relationships play an important role in testing, maintenance and configuration management of software systems. The informal dependency representations fail to observe behavioral connections among subsystems and cause ambiguity in representing different types of dependency relationships. Therefore, dependency in a software system requires a formal and unambiguous representation so that its correct effects can be visualized. In this paper, we present a Colored Petri Net based dependency analysis of a Service Oriented Architecture (SOA) based system that represents specification of dependency relationships and models the dependencies in a SOA based system at conceptual level. Different types of dependency relations are represented in a formal manner by using Service Algebra. A module SOA based system '*Online Bookshop*' has been developed and used for the purpose of modeling and example demonstration. Such modeling can help in identification of inconsistency among services, and web services can be verified for safety and reliability.

ACM CCS (2012) Classification: Information systems → World Wide Web → Web services

Software and its engineering → Software organization and properties → Software system structures → Software system models → Petri nets

Software and its engineering → Software organization and properties → Contextual software domains → Software infrastructure → Middleware

Keywords: Service Oriented Architecture, web services, Petri Net, Colored Petri Net, Service Algebra, dependency analysis

1. Introduction

Software organizations are aware of the fact that without sufficient understanding of the systems they develop, maintenance and evolution become expensive and unpredictable. Service Oriented Architecture (SOA) is an approach to develop software with the assistance of reusable software services. This development paradigm may help to reduce the development time and costs. Understanding dependencies in SOA based systems is essential to performing two functions. The first one is impact analysis, which means understanding which other services are affected when a service becomes unavailable. The second one is service level root cause analysis, which means understanding the cause of a service by looking at the other services it relies on [1].The knowledge of services dependencies is important for a number of management activities. Fault management needs this information to track problems in a distributed service network. Configuration management needs this information to know which services are currently in use and appropriately adapt to changes in the environment. Accounting management needs to know dependencies to appropriately charge for service access. Policy-based management needs to know dependencies and must be able to change them to enforce the policies [2]. All these management activities must have ways to learn the current dependencies, discover their properties, and possibly perform rebinding of services. Adequate dependency analysis guarantees the composability of software services before deploying them and makes complexity manageable. Dependency analysis information is useful for dividing the development of a complex software system into manageable units.

The terms *dependency* and *dependency information* are misunderstood by professionals. Software people have taken this concept in ad-hoc manner and unfortunately dependency information is not always available. It is common for people to experience an installation error or runtime error due to a missing dependency, but

have no-where to find what dependencies are missing. A user may expect dependency information recorded in the documentation distributed along with the software. However, neither the availability nor accuracy of such documentation is guaranteed [3]. Thinking about dependency relationships is about being sensitive to the desirable and undesirable effects of dependencies in various situations. The tasks of unlinking, moving, re-linking in an organizational context are usually expensive and time consuming. They are also fraught with risk, since a neglected dependency could result in an undesirable disturbance to another system when the move occurs. This is why projects are so expensive, and why organizations prefer to work around dependencies rather than address them head-on. In this process, they often create fresh dependencies that can pose an even bigger problem in future. When using Service Oriented Architecture for application development, dependency analysis becomes even more complex. As a result, building a dependency hierarchy between all the components in a SOA becomes extremely difficult, but nevertheless all the more important to ensure that the overall system stays functional while changes are introduced at different levels [4].

In this paper, we present a Colored Petri Net based dependency analysis of a Service Oriented Architecture (SOA) based system that represents specification of dependency relationships and models the dependencies in a SOA based system at conceptual level. Different types of dependency relations are represented in a formal manner by using Service Algebra. A module SOA based system *Online Bookshop* has been developed and used for the purpose of modeling and example demonstration. Such modeling can help in identification of inconsistency among services, and web services can be verified for safety and reliability. The proposed work does not address the definition and internal details of web services. Rather, the main aim of this work is to describe the observable dependencies among web services as a form of logical and temporal relationships.

The rest of the paper is organized as follows. In Section 2, some research efforts, related to the topic, are summarized briefly. In Section 3, a brief introduction of Petri net and service algebra are given. In Section 4, the description of module SOA based system *Online Bookshop* is given, which has been used for modeling and example demonstration. Section 5 describes dependency relationships among web services using Petri nets and service algebra operators. In Section 6, dependency modeling of a SOA based system, through colored Petri net is explained in detail. A comparative study between our approach and other similar approaches are given in Section 7. Finally, we conclude the paper in Section 8.

2. Related Work

The importance of Colored Petri net as a modeling and analysis tool has been recognized by several researchers. Here we limit the discussion only to SOA based systems and especially works related to dependency analysis of SOA based systems.

Valero *et al*. [5], in their paper, have focused on the development of a methodology for the design and validation of composite web services using WS-CDL as the language for describing web services interactions and Petri nets as a formalism that allows to simulate and validate the described systems. They specifically intend, then, to capture timed and prioritized collaborations in composite web services. Zhang *et al*. [6] have introduced a Web Services Net (WS-Net), which is an executable architectural description language incorporating the semantics of Colored Petri Nets with the style and understandability of the Object-Oriented concept and Web services concept. Gehlot *et al*. [3] report on their experience with using Colored Petri Nets (CPNs) for model driven development and quality assessment of a defense-targeted service oriented software architecture. They identified the features of CPN that have resulted in ease of adoption as a modeling tool in 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. Deng *et al*. [7] presented an approach for transforming the composition language WSCI into Colored Petri nets (CPNs) so as to verify the model with existing CPNs-specialized tools. The Colored Petri nets model of a typical use case *Plan and Book Trip* is constructed, analyzed, verified and

simulated as a prototype of WSCI model with the CPNs tools. In his dissertation Wells [8] focuses on the use of colored Petri nets for simulation based performance analysis of industrial sized systems. He has presented an overview of improved facilities for performance analysis using colored Petri nets and a framework for implementing monitoring facilities that observe, inspect, and control simulations of Colored Petri nets (CPNs). Bhubaneswar *et al*. [9] in their paper, deal with the usage of CPNs to verify the semantic web service composition plan represented by Ontology Web Languages for Web Services (OWL-S). The verification is carried out by employing reachability analysis of an algebraic CP-net and simulation using CPN tools. Kolb *et al*. [10] have described orchestration of heterogeneous services in which BPEL and Workflows (WF) are used. They have made effort to convert WFs to Open Workflow nets (oWFNs) so that compiler for executing this net can be mentioned. But their approach is not sufficient to handle dependency issue effectively. Abbassi *et al*. [11] have used a formal model Event-B for web service composition this but approach is not sufficient for dependency analysis.

It can be observed that several works on applications of Colored Petri nets are available in service oriented architecture and other domains. But a specific work of dependency modeling of a SOA based system through Colored Petri nets is rarely available. This work extends the above contributions further by representing dependency relationships, in a SOA based system, through Colored Petri nets which will prove a valued component for validating services at design time.

3. A SOA Based *Online Bookshop* System for Example Demonstration

A module system *Online Bookshop* (using the concept of SOA) has been developed in order to demonstrate dependency relationships among services [12]. The goal of this module system *Online Bookshop* is to take orders from customers and delivers books on time. Payment may be done by cash on delivery/debit card/online banking. To sell books, this program takes an orders from customers, confirms the order, and cancels the order, as suggested by the customer. There are two perspectives in this case, one from customers' point of view and another from sellers' point of view. In the context of customer, the main goal is to buy the appropriate books at least price on right time. In the context of seller, the main goal is to sell more and more books and provide service timely to build goodwill.

Suppose we consider the customers point of view. For purchasing books, there are two situations, either purchasing offline or online. To purchase books offline we have to go to a book shop. Search catalogue to find appropriate books. After finding appropriate books, we need to make payment. Payment can be made by either using cash or debit/credit card. To purchase books online, first a customer visits websites to find appropriate site for purchasing books. When he decides about the book purchase, then he registers himself to the particular website. After registering himself, he logs in to the website. Again he searches for desired books. Desired books are placed in shopping cart and order for the desired books is placed. Appropriate delivery address is provided so that books delivery could be done successfully at right place and to the person. Payment is done using either cash on delivery or debit card or online banking.

Model of *Purchasing Books* has been shown in Figure 1 which shows a model of decomposition of the goal *Purchasing Book*. This goal is decomposed up to manageable sub-goals and a web service is developed for each sub-goal. If a web service does not exist corresponding to a sub-goal, we decompose this sub-goal to further smaller sub-goals so that corresponding to these sub-goals, web services exist. Web services are connected by three types of relationships i.e. sequence, concurrent and exclusion which are denoted by circled arrows, pluses and crosses respectively. In Figure 1, *Search Website*, *Decide a Website*, *Search Books* and *Shopping Cart* sub-goals are sequentially attached. *Order the Book* and *Provide Delivery Address* are performed concurrently. *Payment by Cash* and *Payment by Debit Card* are done exclusively.

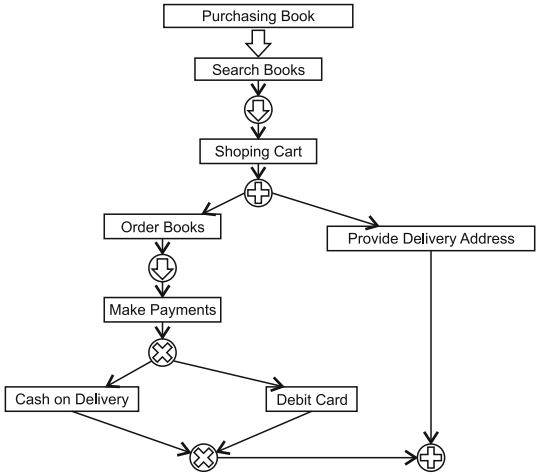


Figure 1. Model of the goal *Purchasing Book Online* [12].

Suppose we take a general book purchase by a client. There are two choices to purchase a book by customer. Either he can go to a book shop and search book catalogue or go online and search books i.e. either he can purchase books online or offline. A goal tree has been drawn for the goal *Purchasing Books* in Figure 2. Figure 2 gives a lot of information how to achieve goal of *Purchasing Books*. To purchase a book is not an automate activity. For achieving this goal, a number of activities have to be performed as is given is Figure 2. In this goal tree there are four levels. Without showing interaction among same level elements, it is not possible to know the relationship among these components. The relationships among same level elements have been depicted by block arrow in Figure 2. There may be three types of relationship: sequence, concurrency and mutual exclusion. Sequence relationships have been depicted by circled arrow, concurrency by circled plus and mutual exclusion by circled cross.

4. Colored Petri Nets (CPNs)

CPN is an established concept for formal modeling of concurrent and distributed system. It is based on functional language standard ML. CPN has capacity of both Petri nets and programming language. Jensen and Kristiansen [13] have defined CPNs formally as follows:

Definition. A CPN is a nine tuple (P, T, A, Σ, V, C, G, E, I), where:

- P is the finite set of places.
- T is the finite set of transitions such that $P \cap T = \phi$.

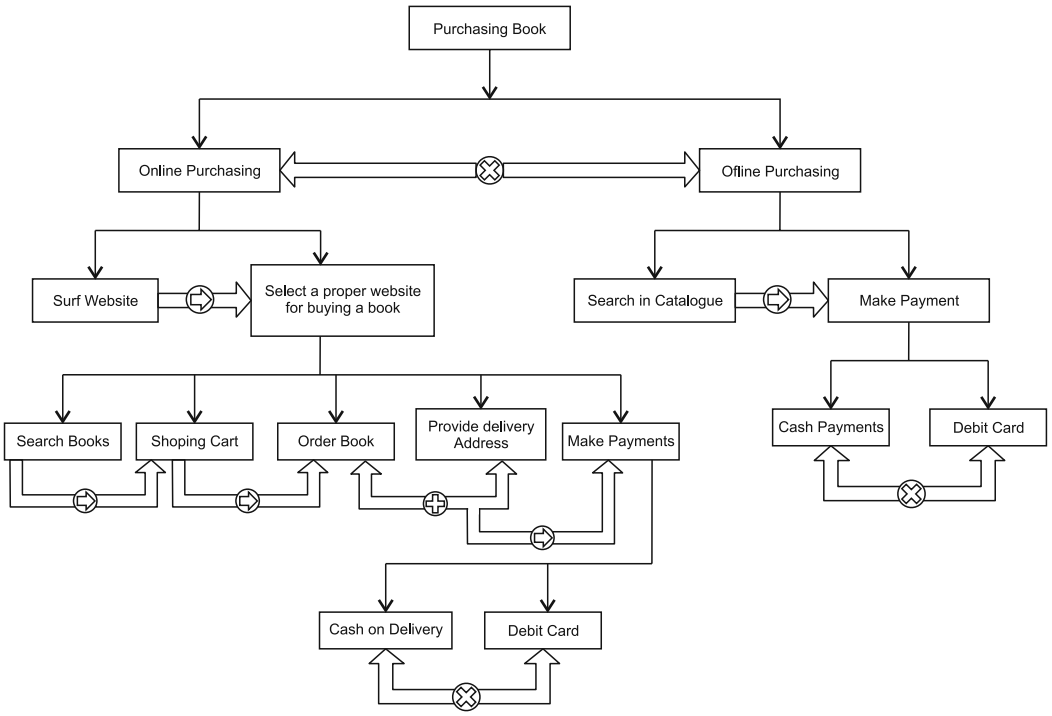


Figure 2. Goal tree for the goal *Purchasing Book*.

- $A \subseteq P \times T \cup T \times P$ is a set of directed arcs.
- Σ is a finite set of non-empty color sets.
- V is a finite set of typed variables such that $Type[v] \in \Sigma$ for all variables $v \in V$.
- $C: P \rightarrow \Sigma$ is a color set function that assigns a color set to each place.
- $G: T \rightarrow EXPR$ is a guard function that assigns a guard to each transition $t \in T$ such that $Type [G(t)] \in Bool$.
- $E: A \rightarrow EXPR$ is an arc expression function that assigns an arc expression to each arc $a \in E$ such that $Type [E (a)] = C(p)MS$, where p is the place connected to the arc a.
- $I: P \rightarrow EXPR$ is an initialization function that assigns an initialization expression to each place p such that $Type [I(p)] = C(p)MS$

By above definition of Colored Petri nets, it is clear that every tuple is formally defined and it becomes unambiguous.

5. Dependency Types Representation Through Petri Nets and Service Algebra

The client-server relationship between multiple pairs of services has to be under constraints of various assumptions that they have regarding each other's structure, functionality and dynamic behavior. Merely stating that a service is dependent on another service is not sufficient. Types of dependency relations along with their critical factors are also important to be explored. This section focuses on formalization of dependency types by means of Petri nets i.e. how different types of dependency relationships, in a SOA based system, can be modeled through Colored Petri nets. We have considered four types of service dependency: control dependency, data dependency, sequence dependency and composition dependency. SOA based system is concurrent, distributed and dynamic by nature. Petri net is suitable modeling approach for this purpose. Petri net model has capacity to handle both static and dynamic dependency and it supports formal and graphical approaches.

In the following subsections, an attempt has been made to model different types of dependency.

5.1. Control Dependency Modeling

A control dependency between two services S_i and S_j specifies the conditions under which service S_j is allowed to enter a state st_j based on the state st_i of service S_i.

Suppose there are two services in *Online Bookshop*, namely *Order Service* and *Payment Service*. We have considered services as black boxes. *Order Service* and *Payment Service* are control dependent. Suppose a customer gives the order for a book. In addition, he gives the information of postal address. *Order Service* sends the control to *Payment Service*. *Payment Service* takes payment from customer. After completing payment, *Payment Service* again sends control to the *Order Service*. Thus *Order Service* and *Payment Service* are control dependent on each other because *Order Service* sends control to *Payment Service* for payment and *Payment Service* after its execution sends control to *Order Service* for completion of order.

We have formalized this dependency using Petri net. In Petri net of *Order Service*, initial state has a token. So when a customer decides to purchase a book, he invokes *Order Service* and gives the mailing address and state t1 fires. After firing transition t1, token goes to the place payment buffer (Pb) and payment service becomes eligible for execution. In this case transition tr1of *Payment Service* is enabled to fire. When *Payment Service* completes its execution, final state fn of *Payment Service* gets a token and transition tk of *Order Service* becomes enable to fire. When final state f of *Order Service* gets token, it means *Order Service* has been executed successfully.

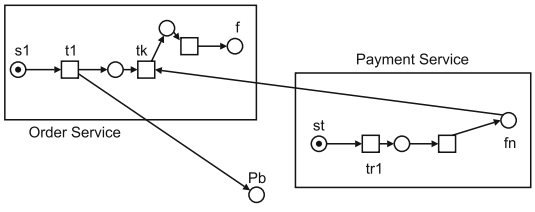


Figure 3. Formal representation of control dependency.

Using the concept of transition firing, it is clear from the diagram that without initiating of *Order Service*, *Payment Service* cannot be initiated for execution.

Formal representation of control dependencies is given here using service algebra. In Figure 3, S1 represents *Order Service* and S2 represents *Payment Service*. Two services are connected to each other by payment buffer service (Pb). Payment buffer (Pb) is a temporary buffer that help to establish communication between *Order Service* and *Payment Service*.

In service algebra, it can be represented as follows:

$$S1 \xrightarrow[t1(via\ Pb)]{} S2$$
$$S1 \xrightarrow[tk(via\ fn)]{} S2$$

The symbol '→' is used for flow operator.

We have used the following symbols as indicated operators:

$\overset{con}{\Rightarrow}$: Control Dependency Operator
$\overset{data}{\Rightarrow}$: Data Dependency Operator
$\overset{com}{\Rightarrow}$: Composite Dependency Operator
$\overset{seq}{\Rightarrow}$: Sequence Dependency Operator

So, control dependency can be represented formally as follows.

$$S1.t1 \overset{con}{\Rightarrow} (via\ Pb)\, S2.tr1 \overset{*}{\rightarrow} S2.fn \overset{con}{\Rightarrow} S1.tk$$

An algorithm for formalization of control dependency is mentioned here. Similar type of algorithms can be written for data dependency, sequence dependency and composite dependency.

Algorithm: Construction of Petri net for control flow dependency

- Model each service using Petri net concepts.
- For each control flow dependency between service Si and Sj add a buffer place bf by which control can flow from service Si to service Sj. This buffer place has input transition in Si and output transition in Sj.
- If control returns from service Sj to service Si, then there is a flow from a place in Sj to a transition in Si.

5.2. Data Dependency Modeling

Data dependency among services can be modeled using Petri net effectively. Suppose there are two services S1 and S2.

Execution of S1 needs some data which can provide service S2? In this case service S1 is data dependent on service S2. Service S1 calls service S2 and service S2 provides information needed by service S1.

Suppose we take an example of *Shopping Kart Service* and *Order Service*. First a customer searches books and he places books in shopping kart. Order is given in shopping kart and information about books is given to *Order Service*. So here *Order Service* is data dependent on *Shopping Kart Service*. It can be represented using Petri net as in the following diagram. From this diagram, it is clear that without providing information from *Shopping Kart Service* to *Order Service*, *Order Service* cannot execute.

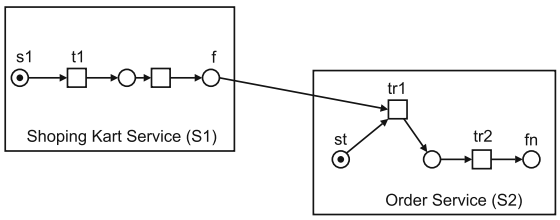


Figure 4. Formal representation of data dependency.

In web service algebra terms, it can be represented as follows:

$$S1.t1 \overset{*}{\rightarrow} S1.f \overset{data}{\Rightarrow} S2.tr1 \overset{*}{\rightarrow} S2.fn$$

It means, first enabling transition is t1 of S1. After a number of firing in S1, token reaches the place f of service net of S1. Now tr1 of service S2 becomes eligible for firing. The resources which need to execute service S2 are available. Service S2 is waiting for data from service S1 which becomes available at this time. After a number of transition firing in service S2, token reaches the place fn of service net of service S2. It means that service S2 has been executed successfully.

5.3. Sequence Dependency Modeling

Sequence dependency among services means execution of services takes place one by one i.e. output of first service is input of second service.

Suppose there are three services S1, S2 and S3. If execution of service S1 is inevitable for initiating the execution of S2 and execution of S2 is essential for initiating the execution of S3, then S3 is sequentially dependent on S2 and S2 is sequentially dependent on S1. This concept can be modeled using service net effectively.

In our example *Online Book Shopping*, first we login, so *Login Service* is called. Then we search books which we have to purchase, so *Search Service* is called. After searching books, we give order for purchase, so *Order Service* is called. In this situation *Order Service* is sequentially dependent on *Search Service* and *Search Service* is sequentially dependent on *Login Service*.

Petri net model of sequential dependency can be described as in the following diagram (Figure 5):

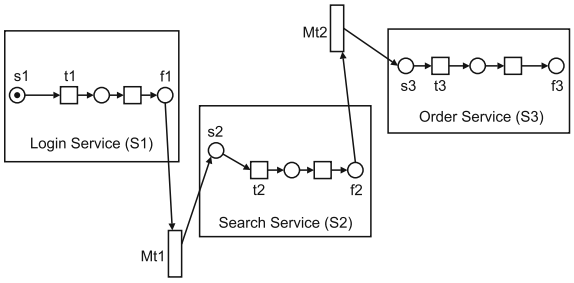


Figure 5. Formal representation of sequence dependency.

In this diagram, there are three services S1, S2 and S3. In this net, the only enabled transition for firing is t1. After completing the execution of service S1, the token goes to place f1. Mediator transition Mt1 plays a role for initiating the execution of service S2. When transition Mt1 fires, then place s2 gets a token and service S2 becomes eligible for initializing execution. After a number of transitions, the token goes to place f2, and mediator transition Mt2 becomes enabled for firing. When Mt2 fires, then initial place s3 of service S3 gets a token and service S3 becomes enabled for initiating the execution. From this diagram, it is clear that S3 is sequentially dependent on service S2, and service S2 is sequentially dependent on service S1.

Sequence dependency can be described using service algebra terms as in the equation below.

$$S1.s1 \xrightarrow{*} S1.f1 \overset{seq}{\Rightarrow} (via\ Mt1)\ S2.s2 \xrightarrow{*} S2.f2 \overset{seq}{\Rightarrow} (via\ Mt2)\ S3.s3 \xrightarrow{*} S3.f3$$

5.4. Composite Dependency Modeling

When a service S is constructed using the services S1, S2, S3 ... Sm, service S is composed of services S1, S2, S3...Sm.

Service S is dependent on services S1, S2, S3 ... Sm. Any defect in any automatic service affects the overall service S. In this way whole SOA based system can be also called a composite service. A SOA based system can be used as a service in a larger project.

This can be represented by the Petri net as follows (Figure 6). There are three services S1, S2

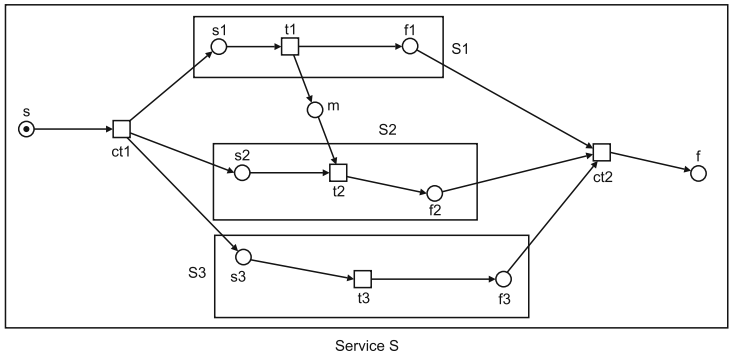


Figure 6. Formal representation of composite dependency.

and S3. Each service S1, S2 and S3 have their own functionality. Service S is composed from services S1, S2 and S3. For demonstration purpose, two special places s and f are added in the figure which is used for starting place and final place respectively. Two special transitions ct1 and ct2 are added to establish the relationship. Initially, token is only in place s and enabled transition is only ct1. When transition ct1 fires, it transfer control to S1, S2 and S3 at the same time. Finally all control comes through transition ct2 and then control transfers to final place f. Service S2 is also dependent on service S1 for the message which is provided by the message place m. Service S will be executed successfully if and only if f gets a token. It is possible only if services S1, S2 and S3 have been executed successfully.

Using service algebra terms it can be shown as in the equation below.

$$S.s\,(via\,ct1) \overset{com}{\Rightarrow} (S1.s1, S2.s2, S3.s3) \overset{*}{\rightarrow} (S1.f1, S2.f2, S3.f3) \overset{com}{\Rightarrow} (via\,ct2)S.f$$

6. Modeling of Dependency Relationships Through Colored Petri Nets (CPN)

In this section, different types of dependency in a SOA based system has been modeled using Colored Petri nets. Conventional Petri net based modeling has few limitations. In Petri net modeling, places have only black tokens i.e. resources cannot be differentiated. Petri Net models have no manipulating power on data. In Petri net model, it is not possible to detect which types of data are stored in places. Abstraction and modular concept is not available in Petri net model. Modeling a SOA based system using Petri net may fail to extract some aspects of dependency relationships. There is no concept of time in classical Petri net. So it does not cover temporal aspect of dependency in SOA based system.

We attempt to model the dependency relationships in a SOA based system with the help of Colored Petri Nets (CPNs). CPN provides a full spectrum of input/output facilities and user defined functions so that one can 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 offers the following four possible analysis approaches: interactive and automatic simulation, performance analysis, state space and invariant analysis and temporal logic based analysis of state space [13]. CPN extends the Petri nets to model both the static and dynamic properties of a system. The graphical part of the CPNs 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 [14]. The main purpose of coloring places is to help people understand the usages 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.

In CPN model, tokens in a place can be distinguished. Every place has its own colored set i.e. data type. By this, it becomes transparent which types of token can reside in a place. CPNs have all the features of classical Petri net in addition to programming features. Due to programming features of CPNs, data flow, and control flow can be manipulated according to requirements. Dependency can be handled effectively using CPN modeling. CPNs has concept of time so it also captures temporal aspects of dependency in SOA based system. Abstraction is the backbone for a SOA based system. Abstraction is achieved in CPN modeling using the concept of hierarchy in CPN. So a complex system can be modeled using hierarchy in a more effective way. Dependency relationships of a complex SOA based system can be understood easily by using CPNs. Thus CPNs modeling provide more generalized view of SOA based system than conventional Petri net modeling.

To represent dependency relationships using CPNs, basically we represent primitive activities of web services and their interactions with other web services. These dependency relations are captured by token firing rules and immediate transitions. We have considered control, data, sequence, composite and hierarchical dependency relationships for the purpose.

6.1. CPN Modeling of Control Dependency

Besides the system variables, there are dependencies between the controls of the components and sometimes, the controls, especially the choices of the paths, rely on both the formal control information and the current system status [15]. A service S2 is control dependent on service S1 if and only if S2's execution is conditionally guarded by S1. For demonstration of control dependence, we have taken two services *Order Service* and *Payment Service*. In the diagram, double line arrows show the control flow between services. *Order Service* sends control for taking payment to *Payment Service*. *Payment Service* takes payment from customer in two ways, either by 'cash on delivery' or by 'debit card'. After taking payment *Payment Service* sends control to *Order Service*. From the CPN diagram of Control Dependency, place *Total Books* has tokens for all books which have been searched by customers. From these books, the customer selects books for purchase. This facility is given by the transition *Book Selected*. When transition *Book Selected* fires, then *Order Service* sends control to the *Payment Service* for making payment and transition *Payment Option* becomes enabled for firing. From the diagram it is clear that after payment confirmation, *Payment Service* sends control to *Order Service* for Order Confirmation.

```
ColsetBNAME = string;
colset BPRICE = real;
colsetOrderCompleted = string;
colsetPaymentType = with CashOnDelivery |
    DebitCard;
colset BNAMEPRICE = product BNAME *
    BPRICE;
colset PTBP = product PaymentType * BPRICE;
colset STRING = string;
var x: BNAME;
var y,z: BPRICE;
val a = Books Delivered;
var p: PaymentType;
var b: STRING;
fun COD(p,y) = if p = CashOnDelivery then 1`y
    else empty;
fun DC(p,y) = if p = DebitCard then 1`(p,y) else
    empty;
```

Figure 7 (a). Declarations of CPN modeling of control dependency.

6.1.1. State Space Analysis for *CPN Modeling of Control Dependency*

A central point in more formal methods is to conduct formal analysis and verification based on a constructed model, i.e. to determine whether or not a system under development or analysis satisfies certain formally stated properties. State space analysis is one of the most prominent approaches for conducting formal analysis and verification. The basic idea of state space is to

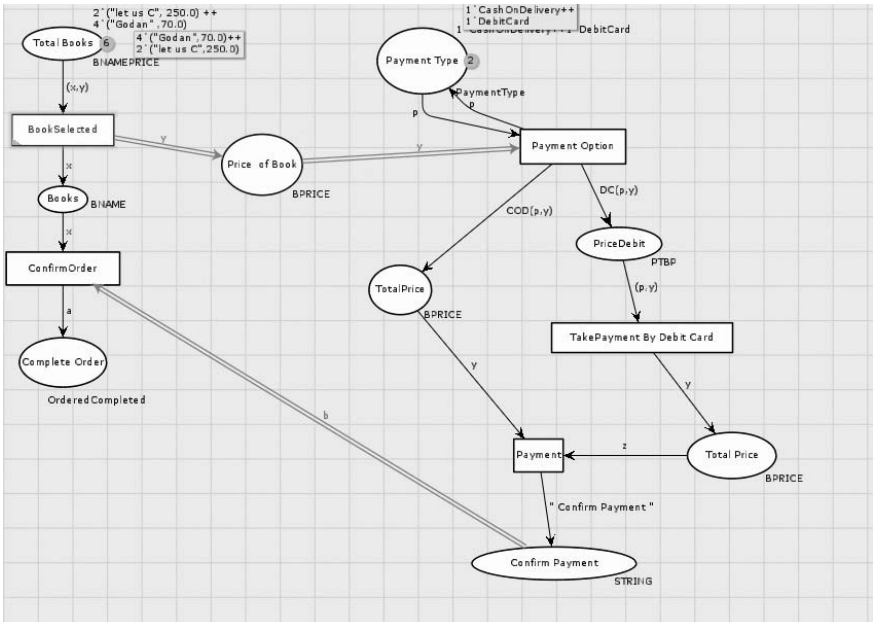


Figure 7 (b). CPN model of control dependency.

calculate all reachable states and state changes of the system and represent these as directed graphs. State space can be constructed automatically with the help of a CPN tool. From a constructed state space it is possible to answer a large set of analysis and verification quantities concerning the behavior of the system such as absence of deadlocks, the possibility of always being able to reach a given state, and the guaranteed delivery of a given service.

The applicability of state space methods is closely tied to the existence of suitable computer tool support and manual calculation, while the inspection of the state space for more than trivial systems is time consuming, error prone and impossible for practical purposes. One of the main advantages of state space methods is that they can provide counter examples i.e. debugging information as to why an expected property does not hold [16].

State space analysis gives important properties for verification and validation of a system design. Some important properties as boundedness, liveness and fairness can be deduced using state space analysis. We have used state space tool of CPN tool software for state space analysis. State space tool can generate reports automatically. On the basis of this report we can observe the control dependency among the services. The state space report on Control Dependency has been shown on Table 1.

Here, we analyze the CPN model of the control dependency. We aim at illustrating the support for state space analysis provided by the tool and the steps provided in conducting the analysis, rather than at giving an exhaustive analysis of the control dependency model.

The first step towards state space analysis of a CP-Net is generation of the state space. This step is fully automatic. The generation statistics for the data dependency model are shown in Table 1. In the CPN modeling of control dependencies there are nine places. The report shows that in all places, upper and lower bound of tokens (maximum and minimum number of tokens) in any reachability marking are mentioned. Home property explains that control starts from a service and after passing through some intermediate services, again it reaches to originating position. Home properties of state space analysis show that there is no home marking in the CPN modeling of Control Dependency i.e. from initial marking to any reachable marking no cycle

Table 1. State space analysis report for *Control Dependency Modeling*.

Statistics

State Space

Nodes	2771
Arcs	9835
Secs	3
Status	Full

Scc Graph (Strongly Connected Components of State Space)

Nodes	2771
Arcs	9835
Secs	0

BoundednessProperties

Best Integer Bounds

Places	Upper Bound	Lower Bound
Books	6	0
Complete Order	3	0
Payment Type	2	2
PriceDebit	6	0
Price_of_Book	6	0
TotalPrice	6	0
Total_Books	6	0
Total_PricebyDB	6	0
Confirm_Payment	3	0

Home Properties

Home Markings

None

Liveness Properties

Dead Markings

35 [2771, 2770,2769,2768,2767...]

Dead Transition Instances

None

Live Transition Instances

None

Fairness Properties

No infinite occurrence sequences

is formed. Liveness property plays a major role in the analysis of any system. Liveness property may indicate about the presence of deadlock in the system. Reports show that reachability graph of CPN model has 35 dead markings i.e. after reaching that marking there is no enabled transition to fire. There is no live transition in CPN modeling of Control Dependency i.e. that transition is always enabled to fire. By the liveness property of CPN modeling, control dependencies among services can be extracted.

6.2. CPN Modeling of Data Dependency

The general purpose of the software system is data manipulation, which means to compute the new system status according to existing data, which we call dependency of data. Once the existing system states (called variables) contain faults, the faults could spread to other variables or other components. How the faults spread depends on how the new variables are manipulated from the existing ones. So it is necessary to define the data dependency to specify the way of fault spreading [15].

Service S2 is data dependent on service S1 if and only if service S2 needs some data form service S1 for full execution of service S2. In CPN modeling of Data Dependency, we have modeled two services *ShoppingCart* and *Order* using CPN tools as in Figure 3. The bold arrow shows the transfer of data from *ShopingCart* service to *Order* service. *ShoppingCart* service has two places and one transition: *Select Books,* and *Order* service has four places and two transitions: *Order Confirm* and *Make Payment*. Transition *Order* works as the mediator for data flow from service *ShoppingCart* to service *Order*. In Figure 8, the only enabled transition to fire is *Select Books*. By firing this transition one by one book is selected and removed from place *Total Books in Search* and put into the place *Books for Order* as list. Transition *Order* is enabled to fire if and only if place *Books for Order* has non empty list of books. Transition *Order Confirm* is enabled to fire if and only if some data from *ShoppingCart* service is available to the place *Ordered Books*. When transition *Order Confirm* fires then variables r and p take data of books list and delivery type respectively from input place and put data as a pair of r and p to the place *BooksDeliveryType*. After firing transition *Make Payment*, order is completed.

```
colset STRING = string;
colset BS = list STRING;
    (*Books Selected for Order)
colset BDT = product BS * STRING;
    (*Books and Delivery type)
val TotalBooks = 3`"Let Us C" + + 2`"Software
    Engineering" + + 5` "Automata Theory";
val DT = 1`"Cash On Delivery" ++1` "Debit
    Card";
var p: STRING;
var q,r: BS;
```

Figure 8 (a). Declarations of CPN modeling of data dependency.

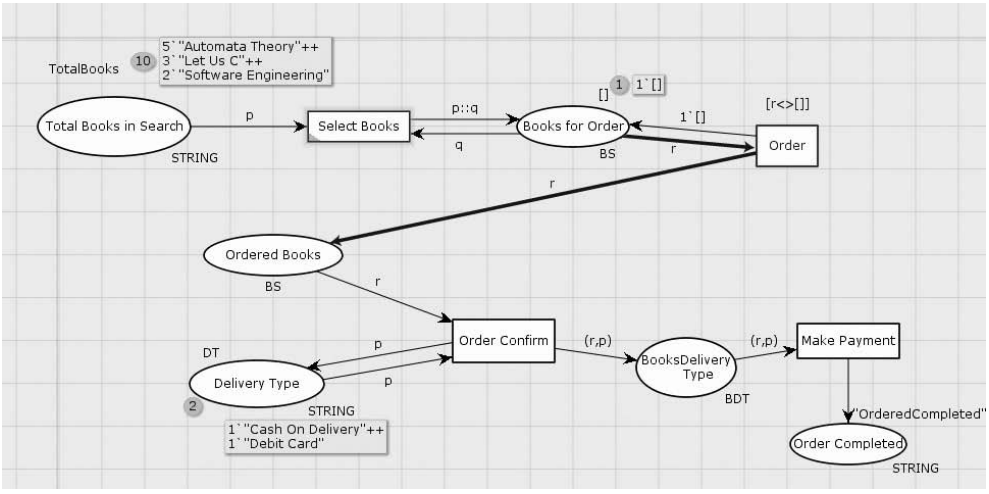


Figure 8 (b). CPN model of data dependency.

6.3. CPN Modeling of Composite Dependency

In this section, we analyze the CPN model of the composite dependency. We aim at illustrating the support for state space analysis provided by the tool and the steps provided in conducting the analysis, rather than at giving an exhaustive analysis of the composite dependency model.

Composition of services play a vital role in a SOA based system. Composition may be static or dynamic. Static composition is performed before compilation of a service. Dynamic composition occurs at run time according to requirement of the requester of the service. Composite service is dependent on all atomic services of which this service has been composed. We have explained composite dependency using an example of *Search* service. *Search* service is composed of three services *SearchByAuthor*, *SearchByName* and *SearchByISBN* services. Composition may be of different types as concurrency, mutual exclusion, etc. We have used the composition of mutual exclusion. In *Search* service, searching of books is done using book name, isbn and author. In Figure 9, CPN model of composite dependency has one place *SearchOption*. There are three functions attached to the out arcs from the transition *SelectSearchOption* in which one will give the output value x and the other will give output *empty* which depends on the value of x. Place *ListOfBooks* provides the book to the transitions *SearchByAuthor*, *SearchByISBN* and *SearchByName*. In this model composite dependency is that only one transition from *SearchByAuthor*, *SearchbyName* and *SearchByISBN* will be fired at a time. Finally token *SearchCompletion* goes to the place *G* after completion of *Search* service.

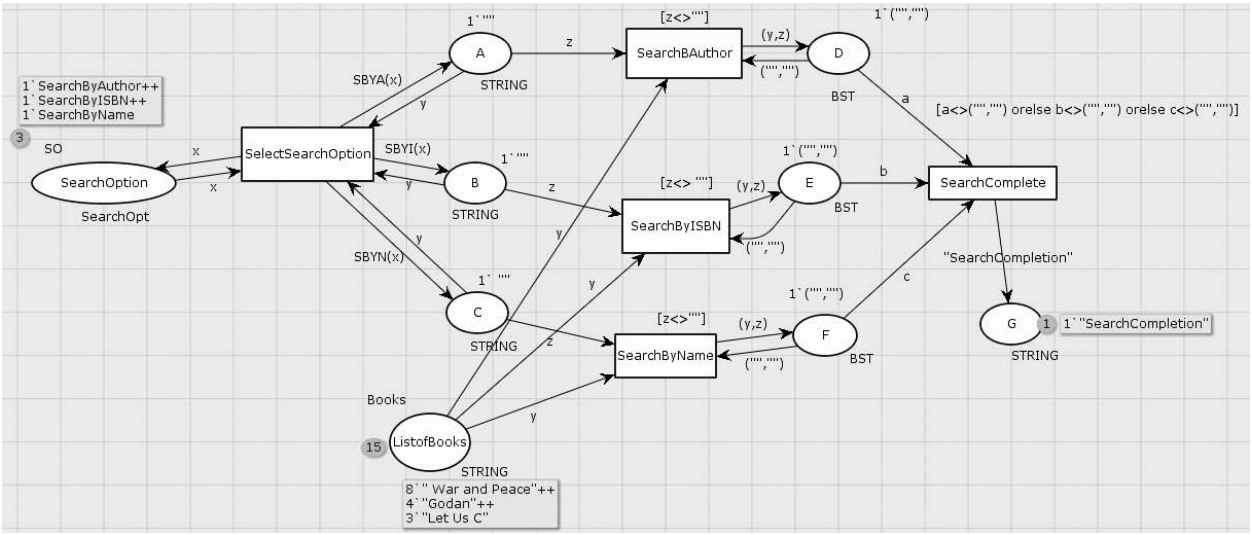


Figure 9 (a). CPN model of composite dependency.

```
colset STRING = string;
colsetSearchOpt = with SearchByAuthor|SearchByISBN|SearchByName;
colset BST= product STRING*STRING;                    (* BST means Books and their search type *)
var a,b,c: BST;
val SO = 1`SearchByName ++ 1` SearchByAuthor ++ 1`SearchByISBN;
val Books = 4`"Godan"++ 4`"Let Us C" ++ 8`" War and Peace";
var x: SearchOpt;
var y,z: STRING;
fun SBYA(x ) = if x = SearchByAuthor then 1`" SearchByAuthor" else empty; (* Search by Author Service*)
fun SBYI(x) = if x =SearchByISBN then 1`"SearchByISBN"else empty; (*Search By ISBN service *)
fun SBYN(x) = if x =SearchByName then 1`"SearchByName"else empty; (* Search By Name Service *)
```

Figure 9 (b). Declarations for CPN modeling of composite dependency

6.4. CPN Modeling of Sequence Dependency

Sequence dependency means that two or more services should execute sequentially i.e. these services cannot execute concurrently. For demonstration of sequence dependency, we have modeled three services, namely *Login* service, *Search* service and *Payment* service using CPN tools as depicted in Figure 10. In this diagram sequence dependency is shown in bold line arrow. It is clear that *Login* service execution is essential for execution of *Search* service and execution of *Search* service is inevitable for execution of *Payment* service. Thus, *Search* service is sequentially dependent on *Login* service and *Payment* service is sequentially dependent on *Search* service.

6.5. CPN Modeling of Hierarchical Dependency

In this section, we analyze the CPN model of the hierarchical dependency model. We aim at illustrating the support for state space analysis provided by the tool and the steps provided in conducting the analysis, rather than at giving an exhaustive analysis of the hierarchical dependency model.

There is a powerful concept of hierarchy in CPN. Hierarchical dependency among the services means dependency between two services which are connected by socket. In Figure 11 there are three figures which represent overall picture of hierarchy. In Figure 11(c) there are two substitution transitions *Order* and *Payment* which abstracts the details of *Order* service and *Payment* service. In this diagram input and output places from substitution transitions are said

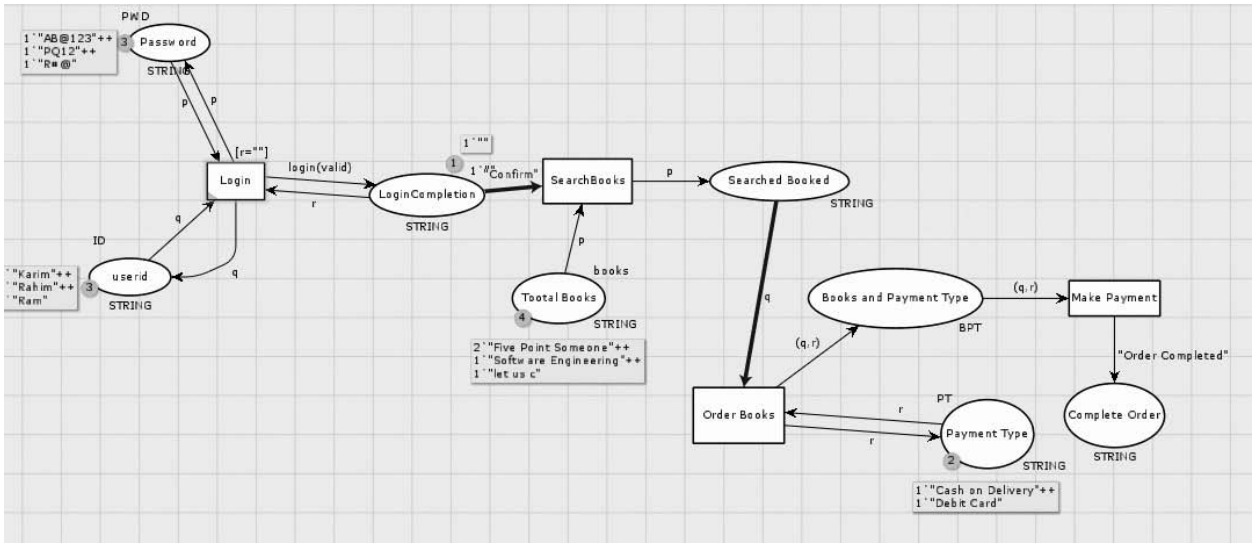
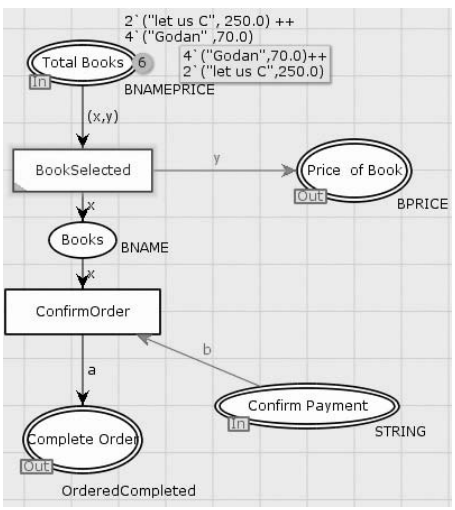


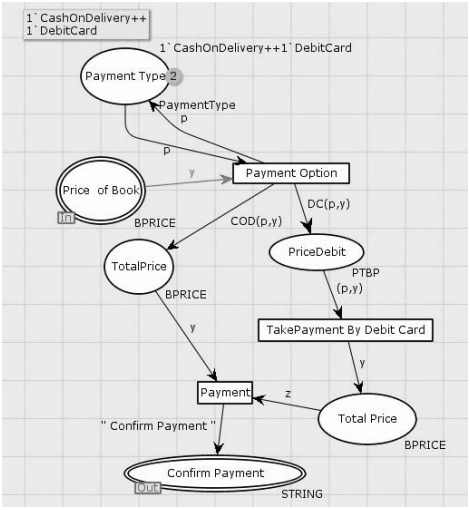
Figure 10 (a). CPN model of sequence dependency.

```
colset STRING = string;
colset BOOL = bool;
var valid:BOOL;
var p,q,r :STRING;
val ID = 1`"Ram"++ 1`"Rahim" ++1`"Karim";
val PWD = 1` "AB@123"++ 1`"R#@"++1`"PQ12";
fun login(valid)= if valid then 1` "Confirm" else 1`"";
val books =1` "let us c"++2` "Five Point Someone"++1`"Software Engineering";
val PT = 1` "Cash on Delivery"++ 1`"Debit Card"; (* PT is used for Payment Type *)
colset BPT = product STRING*STRING;          (* BPT is used for Books and Payment Type Both *)
```

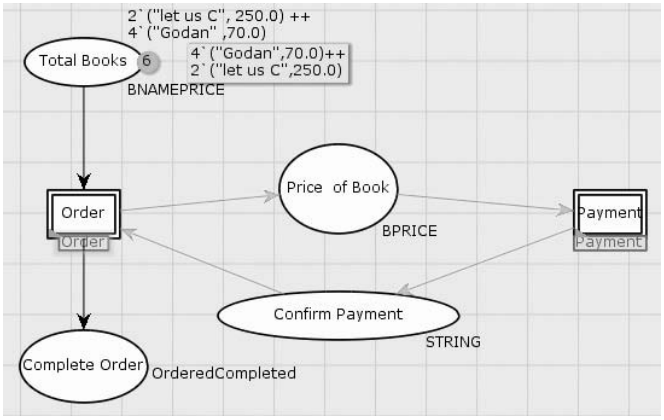
Figure 10 (b). Declarations for CPN modeling of sequence dependency



(a)



(b)



(c)

Figure 11. CPN model of hierarchical dependency.
(a) Order service.
(b) Payment service.
(c) Hierarchical dependency model.

```
colset BNAME = string;
colset BPRICE = real;
colsetOrderCompleted = string;
colsetPaymentType = with CashOnDelivery | DebitCard;
colset BNAMEPRICE = product BNAME * BPRICE;
colset PTBP = product PaymentType * BPRICE;
colset STRING = string;
var x: BNAME;
var y,z: BPRICE;
val a = “Books Delivered”;
var p: PaymentType;
var b: STRING;
fun COD(p,y) = if p = CashOnDelivery then 1`y else empty;
fun DC(p,y) = if p = DebitCard then 1`(p,y) else empty;
```

Figure 11 (d). Declaration of CPN modeling of hierarchical dependency.

to be input, output or input/output sockets. Here transition *Order* sends the control to transition *Payment* through the socket *Price of Book* and gets control from the transition 'payment' through the socket *Confirm Payment*. Details of substitution transitions *order* and *payment* are depicted in Figure 11(a) and 11(b). Input / output socket of substitution transition is said to be input port, output port or input/output port in the expansion of substitution transition. As in Figure 11(a), places *Total Books* and *Confirm Payment* are input ports and places *Price of Book* and *Complete Order* are output ports, which is clear from the diagram. Port places are represented by concentric ellipse.

7. Comparative Study

The use of formal language to demonstrate dependencies between web services helps in validation of systems. Our work is based on service algebra and colored Petri net based representations of dependencies between web services. We compare our work with other representations techniques in web services. For comparison purpose, we have taken research work by Zhang *et al*. [6], Sangal *et al*. [17] and three approaches for web service choreography (composition) as Web Service Choreography Interface (WSCI) [18], Web Services-Choreography Description Language (WS-CDL) [19] and Ontology Web Languages for Web Services (OWL-S) [20]. Zhang *et al*. [6] have discussed WS-Net for specification of services. Sangal *et al*. [17] have given Lattix Inc's Dependency Manager (LDM) which is based on Dependency Structure Matrix (DSM) [17].

Some observed differences are mentioned below:

- The one major difference we observe is that the above mentioned representations are focused on the composition of web services but our work is focused on dependency identification and representation among web services.
- The dependency relationships discussed in above contributions have given equal treatment. In our contribution, we clearly differentiate between data dependency, control dependency, temporal dependency and sequential dependency relationships.
- Our work needs less code to express dependencies between web services.
- WS-Net, an extension of Petri net has been used for web service specification at architecture level. For this purpose, WS-Net has been divided in three layers: Interface Net, Interconnection Net and Interoperation Net. In our work, we combine all into single entity that performs all three functionalities.
- WS-net describes interface net as the most abstract form of services in which only input port and output port are visible i.e. it is just like black box for a service. Interaction net is described for interaction between services and interoperation net shows the internal details and dynamic behavior of a service. In our research, colored Petri net has been used for simple service and for abstraction purpose hierarchical concept of CPNs has been used. To abstract the properties of a service, substitution transition method has been used.
- We have identified different types of dependencies observable in SOA based systems and formalized each type of dependency with service net.
- LDM takes software code as input and produces architecture specification of the system. But in our research work, dependency analysis has been done at analysis and design levels and efforts have been made to construct dependency model using CPN. This model can be used by various stake holders during system development and implementation.
- All representations have basic mechanism to describe message exchange among web services. One commonality in our work and above mentioned works is that all representations depict the structural and behavioral analyses among web services.

8. Conclusion

In this paper we present a Petri net and service algebra based formal approach for modeling and analyzing dependence patterns among web services. Unlike informal modeling, a formal model furnishes a basis for formal validation of

web services that enables one to develop reliable systems. The proposed approach is strong enough to analyze simple and complex web services and their relationships at design level. As a result, the dependency modeling enhances the understandability of web service oriented applications.

References

[1] S. Basu *et. al.*, "Toward web service dependency discovery for SOA management", in *IEEE International Conference on Services Computing*, pp. 422–429, 2008. http://dx.doi.org/10.1109/scc.2008.45

[2] J. Zhou *et al.*, "Dependency-aware Service Oriented Architecture and Service Composition " in *Proceeding of IEEE International Conference on Web Services*, Salt Lake City, UT 2007 pp. 1146–1149, 2007. http://dx.doi.org/10.1109/ICWS.2007.71

[3] V. Gehlot *et al.*, "Model driven development of a service oriented architecture (SOA) using colored Petri nets", in *Workshop on Quality in Modeling*, pp. 63–76, 2006.

[4] A. Seth, "Challenges of Performing Software Configuration Management in a Service Oriented Architecture", White Paper, 2008.

[5] V. Valero *et al.*, "A Petri net approach for the design and analysis of Web Services Choreographies", The Journal of Logic and Algebraic Programming, vol. 78, pp. 359–380, 2009. http://dx.doi.org/10.1016/j.jlap.2008.09.002

[6] J. Zhang *et al.*, "WS-Net: A Petri-net Based Specification Model for Web Services", in *IEEE International Conference on Web Services*, pp. 420–427, 2004. http://dx.doi.org/10.1109/icws.2004.1314766

[7] X. Deng *et al.*, "Modeling and Verifying Web Service Composition Using Colored Petri Nets Based on WSCI", in *IEEE International Conference on Industrial Engineering and Engineering Management*, Singapur, 2007, pp. 1863–1867, 2007.

[8] L. Wells, "Performance Analysis Using Coloured Petri Nets", PhD Dissertation, CPN Centre, Dept. of Computer Science, University of Aarhus, 2002.

[9] A. Bhuvaneswari *et al.*, "Assessment of Service Composition Plan Using Colored Petri Nets", *International Journal of Engineering and Computer Science*, vol. 1, pp. 3736–3742, 2014.

[10] S. Kolb *et al.*, "Bridging the heterogeneity of orchestrations - A Petri net-based integration of BPEL and Windows Workflow 2012", in *IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 1–8, 2012.

[11] I. Abbassi *et al.*, "An Event-B Driven Approach For Ensuring Reliable and Flexible Service Composition", *International Journal of Services Computing*, vol. 2, pp. 45–57, 2014.

[12] Ratneshwer and P. Kumar, "Dependency analysis of a SOA-based system through Petri nets and service algebra", *Int. J. Software Engineering, Technology and Applications, Inderscience*, vol. 1, pp. 172–189, 2015.

[13] K. Jensen and L. M. Kristensen, "Coloured Petri Nets: Modelling and Validation of Concurrent Systems", Springer-Verlag Berlin Heidelberg: Springer, 2009. http://dx.doi.org/10.1007/b95112

[14] K. Jensen, "Coloured Petri Nets: a high level language for system design and analysis", in *Advances in Petri Nets*, New York, NY, USA, pp. 342–416, 1990.

[15] Y. Li, "Diagnosis of Large Software Systems Based on Colored Petri Nets", PhD Dissertation, University of Paris, 2010.

[16] L. M. Kristensen, "State Space Methods for Coloured Petri Nets", Ph.D Dissertation, University of Aarhus, Denmark, 2000.

[17] N. Sangal *et al.*, "Using dependency models to manage complex software architecture", in *20th annual ACM SIGPLAN conference on object oriented programming systems languages and applications*, pp. 167–176, 2005. http://dx.doi.org/10.1145/1103845.1094824

[18] A. Arkin *et al.*, "Web Service Choreography Interface (WSCI) 1.0", Technical Report, W3C, 2002.

[19] N. Kavantzas *et al.*, "Web Services Choreography Description Language Version 1.0", Technical Report, W3C, 2005.

[20] G. Antoniou and F. van Harmelan, "A Semantic Web Primer", MIT Press second edition, 2008.

Received: September 2015
Revised: November 2015
Accepted: November 2015

Contact addresses:
Pawan Kumar
Department of Computer Science (MMV)
Banaras Hindu University, India
Varanasi-221005
e-mail: pawan.bhuphd@gmail.com

Ratneshwer Gupta
Department of Computer Science (MMV)
Banaras Hindu University, India
Varanasi-221005
e-mail: ratnesh@bhu.ac.in

Mr. Pawan Kumar is working as a Senior Research Fellow in the Department of Computer Science (MMV), Banaras Hindu University, India. His research area is dependency analysis of SOA based systems. He is pursuing his doctoral work under supervision of Dr. Ratneshwer.

Dr. Ratneshwer received his Ph.D. in Component Based Software Engineering from Indian Institute of Technology, Banaras Hindu University, Varanasi (IIT-BHU), India. His research area is CBSE and SOA. He is serving as an Assistant Professor in the Department of Computer Science (MMV), Banaras Hindu University, India. He has been actively involved in taeching and research for the last 8 years. His research monograph has been published by LAP Germany and one book chapter by IGI Global Publication. He has 16 research papers published in International journals and 16 research papers in international/national conference proceedings.