Modelling Co-occurring Changes in a BPEL Process with Petri Nets

Parimala N.
Department of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, India

Keywords: SOA, Service, Petri Net, Co-occurring Changes.

Abstract: Changes to a BPEL process can occur due to changes in its requirements. In this paper, we are considered with more than one change which must be made simultaneously. Such changes are referred to as co-occurring changes. Petri nets are used to express the changes because of their applicability to reflect changes to the system as an evolution of the Petri net model. Each change is expressed as a rewrite rule. The rewrite rules are analysed to determine the order in which they have to be applied. The rules may have to be executed in parallel or in a particular order. A change model, L-Change is defined which enforces that either both changes take place or none.

1 INTRODUCTION

The creation of services and the manner in which services are used or interact with each other is specified by Service Oriented Architecture (SOA) (Erl, 2005). The specification or the interface of a service is separated from its implementation (Zhang, 2006) (Laskey and Laskey, 2009). When an application is large and complex, a single service is not very useful. Instead, many services are assembled together into a composite service in order to realize the complex computation (Erl, 2005) (Newcomer and Lomow, 2005). Composition is traditionally accomplished either as an orchestration or a choreography. In orchestration, there is a single process which realizes the application. However, the business process uses the functionalities of other services in order to achieve the complex functionality. Web Service Business Process Execution Language (WS-BPEL) is the de facto language to express business processes which are based on web services (Barreto et al., 2007). A composite service expressed using WS-BPEL is also known as a WS-BPEL process. On the other hand, in choreography, web services interact with each other to realize a complex application. There is no central process which implements the invocation of web services. Instead, all the web services are in a peer-to-peer relationship. They interact with each other to achieve a common business goal. Choreography can be specified using Web service Choreography Description Language (WS-CDL) (Ross-Talbot and Fletcher, 2006).

Services undergo changes. Normally, this is either due to changes in the business process itself or because partner services have undergone a change. In this paper, we are concerned with changes in the business process itself. These changes come about due to changes in the business policies. The modification to the process itself may be a single change or it could be a couple of changes. Consider an example of a business process which captures granting leave to an employee. There could be a change in the business policy wherein the policy 'leave can be granted anytime' has now changed to 'leave can be granted if the employee has not availed of leave in the last week'. Such a change is termed as a 'single change'. There can be more than one change to a business process. These changes can be independent changes. For example, the policy 'an employee can take 5 days of leave at a time' is changed to 'an employee can take 10 days of leave at a time' is independent of the previous policy. We are concerned with policies which are not independent. In particular, the changes to the business process dictated by the policies must co-occur. Consider an example of a hospital where patients seek an appointment to see a doctor. Let the original policies be that 'a doctor sees a patient normally for 20 minutes' and 'the appointment slot for a patient is 20 minutes'. Let us say that there is a change in the hospital policies. 'A doctor sees a patient for 15 minutes' and 'the appointment slot for a patient is 15

N., P.
Modelling Co-occurring Changes in a BPEL Process with Petri Nets.
DOI: 10.5220/0007741804100416
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 410-416
ISBN: 978-989-758-375-9

minutes'. Both the changes must co-occur for a consistent system. In this paper, we are concerned with these co-occurring changes.

Petri nets (Petri, 1962) have been applied in many different contexts (Murata, 1989). Many concurrent and discrete event distributed systems have been modelled using Petri nets (Gracanin et al., 1993) (Dam and Ghose, 2015) (Kristensen et al., 1998) (Iordache and Moldoveanu, 2014). Petri nets have also been used to express work flow of a process (Van der Aalst, 1998) (Adam et al., 1998) (Gou et al., 2000) (Hamadi and Benatallah, 2003). Using Petri nets it is possible to verify and check the composition of processes, the soundness and other properties (Aalst, 1997) (Hinz et al., 2005). More specifically, in this paper, we model the changes using reconfigurable Petri nets. We adopt the model for reconfigurable nets as a system of rewriting rules as given in (Llorens and Oliver, 2004). Here, the system configuration is defined as a Petri net and a change in configuration is described as a graph rewriting rule.

We adopt Petri nets to model changes because of their applicability to reflect changes to the system as an evolution of the Petri net model. Reconfigurable Petri nets modify the structure of the net by replacing one subnet with another. The replacement is defined by a rewriting rule which replaces places, transitions and tokens of a subnet with those of the other replacing subnet. The co-occurring changes are expressed as two rewriting rules and both the rules are enforced. That is, the new configuration is arrived at by applying graph modification for both the rules. While doing so, we examine whether the rules can be applied in any order, or have to be sequential or have to be executed in parallel. A change model, L-Change, is defined wherein the co-occurring changes are expressed.

The rest of the paper is organized as follows. Section 2 puts the work in context by comparing it with related work. Section 3 explains the example which is used in this paper. Section 4 explains the manner in which co-occurring changes can be expressed as rewriting rules of a reconfigurable Petri net. The change model is given in Section 5. An examples is considered in section 6. Section 7 is the concluding section.

2 RELATED WORK

In service oriented computing, a composite service invokes other web services in order to fulfill a task. There exists dependencies between the provider services and the receiver service wherein the provider services are a part of the composition. Whenever the business process changes, invoked services may have to change. Similarly, when the service provider makes a change, it may impact the business process (Wang et al., 2012). In (Novotny et al., 2013), the dependency between a service which invokes another service to avail of its functionality is termed as Inter-dependency between services. The interdependency give rise to co-changes, wherein more than one service undergoes changes simultaneously. Mining Software Repositories log the different versions of the services. These versions are studied to identify the services which changed at the same time and infer, if there is dependency between them. Mining techniques have been used to study co-change dependency between services (Zimmermann et al., 2005) (Dam and Ghose, 2015) (Li et al., 2013).

When changes takes place across different services, then consistency has to be maintained. When changes take place and the system of services is inconsistent, then, additional changes have to be made to bring the system of services to a consistent state. Change propagation deals with identifying the additional changes that are needed after the primary or main changes are made (Dam and Ghose, 2015). The emphasis is on studying the impact of changes. One of its main objectives is consistency preservation across services. In (Zhang et al., 2014), dependencies between activities are defined at the requirements level. When the changes are propagated for consistency, the propagation is analyzed by classifying the propagated changes further as direct and indirect propagation.

The work proposed in this paper differs from earlier work in two ways:

a) Co-occurring changes defined here is concerned with more than one change in a single process whereas co-change deals with the interdependency between services.
b) The proposed change model permits none or both the changes to be effected so that consistency is maintained whereas change propagation deals with additional changes that have to be made to maintain a consistent system.

3 A HOSPITAL EXAMPLE

To illustrate this work, an application from a Hospital out-patient department is considered. The application deals with appointments with a doctor. Let us assume that a patient has to take an appointment before consulting a doctor. The goal of the Doctor

Appointment System (DAS) is to provide the desired appointment with a doctor. While doing so, DAS has to ensure that rules of the hospital regarding appointments are enforced. The patient has to take an appointment, pay the consultation fees and then see the doctor. The following steps are performed for an appointment:

1. Receive request from user
2. Check the Doctor availability.
3. Allot a slot
4. Take payment
5. Generate appointment slip.

The process of seeking an appointment is expressed as a composite process using orchestration. The process is shown in Figure 1.

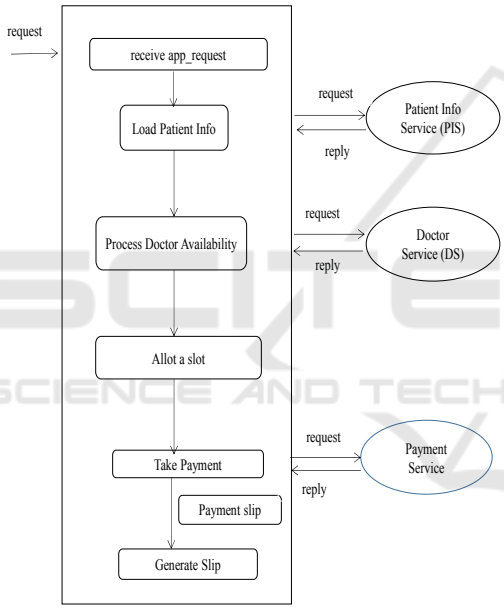

Figure 1: Doctor Appointment System (DAS).

A BPEL process models the workflow of a process but at an abstract level. We represent a BPEL process as a Petri net (Verbeek and van der Aalst, 2005). We model each service of DAS as a place and the invocation of member services is represented by a Petri net transition. The Petri net corresponding to Figure 1 is given in Figure 2. In this work flow, when a new user arrives, a marking is placed in state S_1. The system loads the user information at transition T_1. The token moves to S_2. Transition T_2 is now fired which finds the availability of the doctor. If the doctor is available, a slot is allotted. The token moves to S_5. The transition T_4 is fired and the payment is accepted. The appointment slip is generated by T_5.

4 REWRITING RULES

In this paper, we are concerned with co-occurring changes. Co-occurring changes are defined first followed by the manner in which they are modelled.

Co-occurring Changes: Two changes are said to be co-occurring if the changes are to occur simultaneously.

When both the changes are made, the system is consistent. If only one of the changes is made, then the system is inconsistent.

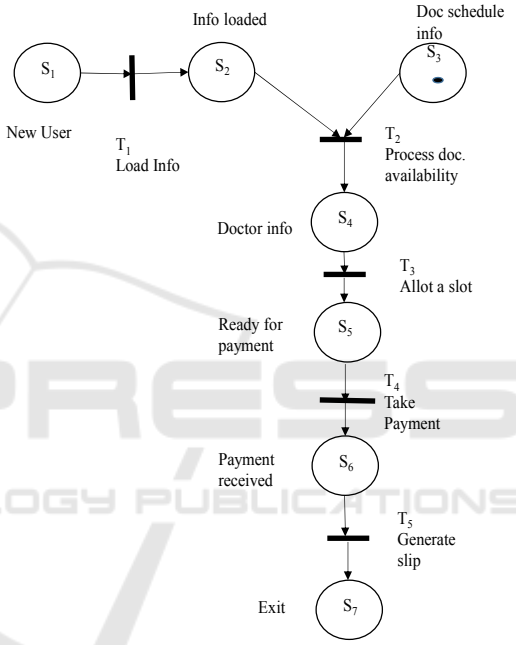

Figure 2: Petri net of DAS.

To put the work in context, an example of co-occurring changes is considered first. In DAS, the services, though independent, do implement certain policies of the Hospital. Let us say that the current policy of the Hospital is that doctors are available for 1 hour daily and the appointment slots are of 20 minutes duration.

The policy is available in state S_3 and implemented in transition T_2. Let us assume that the policy of the hospital undergoes a change. The doctors are now available for 45 minutes and the appointment slots are for 15 minutes. This change has to take place with immediate effect. There are two implications of this change:

a) the schedule of the doctors will change.

b) the patients will get slots of 15 minutes.

Let us first highlight the problems if the change is made in only one place. Consider first the situation where the change is made only in T_2. The patients will continue to get appointment for 20 minutes whereas the doctor is available for 15 minutes. Similar situation arises if the change is made only in T_3. The changes are co-occurring changes to the DAS process. The changes have to be made simultaneously.

4.1 Re-configurable Petri Nets

Reconfigurable Petri nets are an extension of Petri nets. The structure modifying rules replace one subnet by another subnet (Llorens, 2004). The net rewriting rules specify the places and the transitions that are to be rewritten. This, in turn, modifies the flow of tokens in the net. The net is reconfigured according to the rule and is not dependent on the context in which the rewriting takes place.

A net rewriting system was proposed in (Badouel et al., 2003). It is defined as follows.

"A net rewriting system is a structure N = (R, Γ0, M0) where R = {r1, . . . , rh} is a finite set of rewriting rules, Γ0 = (P0, T0, F0) is a Petri net and M0 : P0 → N is a marking associated with Γ0.

A rewriting rule r ∈ R is a structure r = (L, R, τ, •τ, τ•) where:

1) L = (PL, TL, FL) and R = (PR, TR, FR) are Petri nets called the left-hand side and the right-hand side of r, respectively;

2) τ ⊆ (PL ×PR)∪(TL ×TR), called the "transfer relation" of r, is a binary relation relating places of L to places of R and transitions of L to transitions of R (PLτ ⊆ PR, τPR ⊆ PL, TLτ ⊆ TR, τTR ⊆ TL) and

3) •τ ⊆τ , and τ• ⊆τ are sub-relations of the transfer relation called the input interface relation and the output interface relation, respectively."

4.2 Types of Changes

The rewriting rule, as stated above, may introduce places, transitions or both in the right-hand Petri net and then show the relationship between left-hand Petri net and right-hand Petri net. It is also possible that existing states/places of the left-hand Petri net are dropped in the right-hand Petri net. The additions/deletions of places/transitions can occur in both the rules which represent co-occurring changes.

It is relevant to ask the question whether the rewriting rules can be executed in any order. Consider the two rewriting rules

Alter Availability

Alter Appointment

These may have to be executed in parallel. In some other case they may have to be executed sequentially. We examine the rewrite rules more closely below. Towards this, we define the following properties of the rewriting rules for r1 and r2.

4.2.1 Disjoint Rules

Let the original Petri net be Γ0 = (P0, T0, F0). Consider two rewriting rules r1 and r2. Let r1 = (L1, R1, τ1, •τ1, τ1•) and r2= (L2, R2, τ2, •τ2, τ2•). Two rewriting rules r1 and r2 are disjoint if the transfer relations τ1 and τ2 have nothing in common. That is, τ1 ⊆ (PL1×PR1)∪(TL1×TR1) and τ2 ⊆ (PL2 ×PR2)∪(TL2 ×TR2) do not have common places or transitions. In other words,
PL1 ⋂ PL2 = Φ , TL1 ⋂ TL2 = Φ, PR1 ⋂ PR2 = Φ , TR1 ⋂ TR2 = Φ

In this case, r1 and r2 are rewriting different parts of the Petri net Γ0. In this case, the rules can be executed in any order.

4.2.2 Non-Disjoint Rules

Two rewriting rules r1 and r2 are non-disjoint if the transfer relations τ1 and τ2 have something in common. The common part can be a place or a transition. Consider, first, the places. If places are added, then it stands to reason that transitions are always added. Similarly, if places are deleted then transitions emanating from these places will also be deleted.

The non-disjoint rules are analyzed in terms of whether the common place appears in the left-hand side or the right-hand side in each of the transfer relations. Consider the transfer relations τ1 and τ2 of two rules r1 and r2.

{ ({p1, p4}), ({p2}) }⊆ τ1 and { ({p1}), ({p3}) }⊆ τ2

Here, the place p1 appears in the transfer relations of both the rules. If r1 is executed first, then, in the resulting Petri net Γ1 = (P1, T1, F1) the place p1 will not exist and therefore, rewrite rule r2 cannot be executed. Similar is the case if r2 is executed before r1. Thus, the rules r1 and r2 have to be executed in parallel.

The different cases are analysed. The details are not included for the sake of brevity. The non-disjoint rules for changes in places fall into four categories.

a) Rules r1 and r2 are to be executed in parallel
b) Rules r1 and r2 are to be executed in the order r1 r2
c) Rules r1 and r2 are to be executed in the order r2 r1
d) Rules r1 and r2 can be executed in any order

These are referred to Type 1, Type 2, Type3 and Type 4 changes respectively.

Consider, now, the case when only transitions are added/deleted from the existing Petri net. The transfer relations are analysed in terms of whether the common transition appears in the left-hand side or the right-hand side in each of the transfer relations. Consider the transfer relations τ1 and τ2 of two rules r1 and r2.

$\{ (\{t1\}), (\{t2\}) \} \subseteq \tau1$ and $\{ (\{t3\}), (\{t1\}) \} \subseteq \tau2$

Here, the transition, t1 is replaced in τ1 and re-introduced in τ2. These rules have to be executed sequentially. Specifically, r1 has to be executed first followed by r2. If the order is reversed, then the common transition ({t1} in the example above) will not exist in the resultant Petri net.

The different cases are analysed. The details are not included for the sake of brevity. As in the case of places, the non-disjoint rules for changes in transitions also fall into four categories.

a) Rules r1 and r2 are to be executed in parallel
b) Rules r1 and r2 are to be executed in the order r1 r2
c) Rules r1 and r2 are to be executed in the order r2 r1
d) Rules r1 and r2 can be executed in any order

These are referred to as Type 5, Type 6, Type 7 and Type 8 changes respectively.

5 MODELLING CO-OCCURRING CHANGES

The changes that can occur in a system as explained in section 4 is given in Table 1.

We define the change model by introducing L-Change which is defined as follows:

Definition: L-Change is a Petri net {W,R, S, i, o } where

- W is a finite set of places representing the states of a composite service
- R is a fine set of transitions representing co-occurring changes for a composite service
- $S \subseteq (W \times R) \cup (R \times W)$ is a set of directed arcs representing pre-condition and a post-condition for a change
- i is the input place or the starting place
- o is the output place or the ending place

Table 1: Co-occurring changes.

change	Order
D	r1 r2 or r2 r1
Type 1	r1 || r2
Type 2	r1 r2
Type 3	r2 r1
Type 4	r1 r2 or r2 r1
Type 5	r1 || r2
Type 6	r1 r2
Type 7	r2 r1
Type 8	r1 r2 or r2 r1

Figure 3 models co-occurring changes of a composite service. It consists of ten places and nine transitions. The initial place is CS. This is the initial state of the composite service before any change takes place. It consists of nine tokens. The tokens correspond to the nine different types of changes that can take place. When a change occurs, the corresponding transition is fired. For example, when disjoint changes occur then the corresponding transition is fired and the token moves from CS to CS_D.

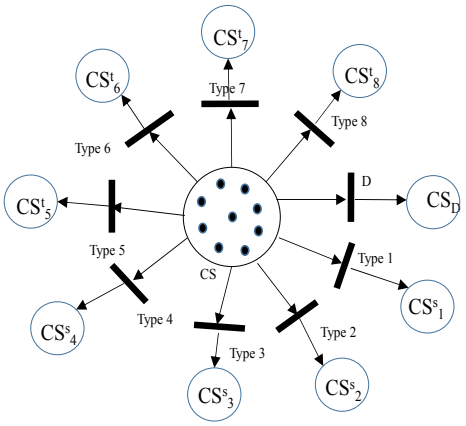

Figure 3: L-Change.

5.1 Modelling Re-writing Rules

Once a token moves to one of the nine places other than CS, then the rewriting rules corresponding to that place have to be executed. The order in which the rewriting rules are to be executed can, again, be expressed as Petri-nets. Consider Type 1 change. Figure 4 represents the Type 1 rule for change in states. CS^s_1 is the starting state of the Type 1 rule. If a token is placed here, then it implies that a change of the form described in Type 1 rule is triggered. The Petri net shows that the rules r1 and r2 have to be executed in parallel. When both the rules are executed, the token moves to CS' which is the new composite service. For a consistent system, the token must be either at the starting or at the ending place, that is, either at CS or CS'.

Similarly, Petri nets for all the nine types of changes are defined. They are not explicitly included for the sake of brevity.

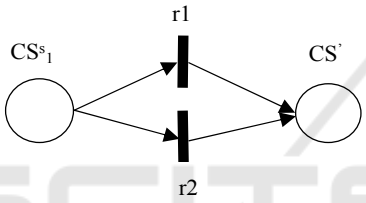

Figure 4: Disjoint rule.

6 CASE STUDY

Consider the example given in section 3. Here, the original policies were that 'a doctor sees a patient for 20 minutes' and 'the appointment slot for a patient is 20 minutes'. Let us say that there is a change in the hospital policies. 'A doctor sees a patient for 15 minutes' and 'the appointment slot for a patient is 15 minutes'. Both the changes must co-occur for a consistent system.

The rewriting rules are

τ_1 = {({S2, S3}, {S2, S3}), ({T2}, {T2'}), ({S4}, {S4'})}
The input interface relation is {({S2, S3}, {S2, S3})}, and the output interface relation is {({S4},{S4'})}

τ_2 = {({S4}, {S4'}), ({T3}, {T3'}), ({S5}, {S5'})}
The input interface relation is {({S4}, {S4'}}, and the output interface relation is {({S5},{S5'})}

Here, the rules have to be executed in parallel. The modified Petri net is given in Figure.5.

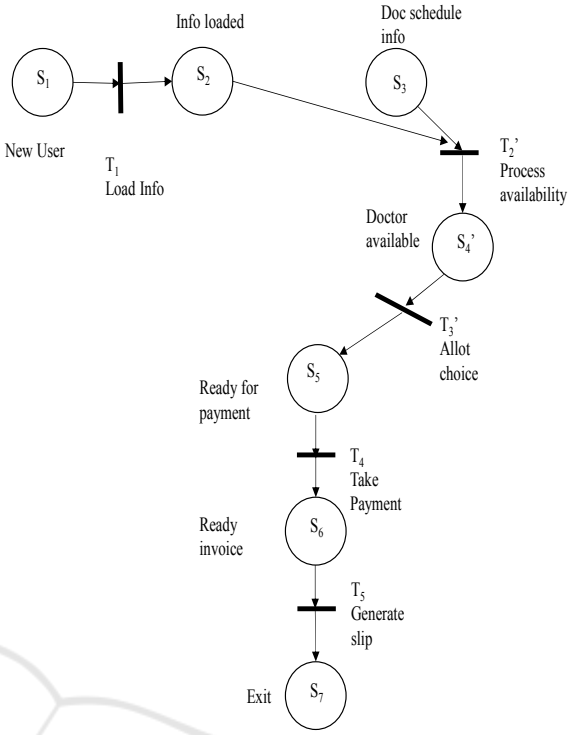

Figure 5: Changes in Appointment Slots.

7 CONCLUSION

In this paper we have considered the issue of enforcement of co-occurring changes. Co-occurring changes are those which must both be effected or none of them. To enforce this, recourse was taken to express the changes as rewriting rules of reconfigurable Petri net. The enforcement of co-occurring changes was expressed as a change model, L-change. The model was a Petri net model.

The types of changes were studied to examine whether the two rewriting rules can be executed in any order. The changes were classified according to the order in which the rewriting rules are to be executed. The order was, again, expressed as Petri net.

The changes are to be effected in a BPEL process. In this paper, the changes were expressed as net rewriting rules. We are working on generating a BPEL process from the Petri net so that the changes can be shown to have been made to the BPEL process itself.

ACKNOWLEDGEMENTS

This work was supported by Jawaharlal Nehru University, New Delhi under University with Potential for Excellence grant, UPE II, Project Id 114. I would also like to thank Mr. Sumit Sharma for suggesting the example.

REFERENCES

Van der Aalst, W. M., 1997, June. Verification of workflow nets. In *International Conference on Application and Theory of Petri Nets* (pp. 407-426). Springer, Berlin, Heidelberg.

Van der Aalst, W. M., 1998. The application of Petri nets to workflow management. *Journal of circuits, systems, and computers*, *8*(01), pp.21-66.

Adam, N. R., Atluri, V. and Huang, W.K., 1998. Modeling and analysis of workflows using Petri nets. *Journal of Intelligent Information Systems*, *10*(2), pp.131-158.

Badouel, E., Llorens, M. and Oliver, J., 2003, June. Modeling Concurrent Systems: Reconfigurable Nets. In *PDPTA* (pp. 1568-1574).

Barreto, C., Bullard, V., Erl, T., Evdemon, J., Jordan, D., Kand, K., Knig, D., Moser, S., Stout, R., Ten-Hove, R. and Trickovic, I., 2007. Web services business process execution language version 2.0. *Specification, OASIS*.

Dam, H. K. and Ghose, A., 2015. Mining version histories for change impact analysis in business process model repositories. *Computers in Industry*, *67*, pp.72-85.

Erl, T., 2005. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India.

Gou, H., Huang, B., Liu, W., Ren, S. and Li, Y., 2000. Petri-net-based business process modeling for virtual enterprises. In *Smc 2000 conference proceedings. 2000 IEEE international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0* (Vol. 5, pp. 3183-3188). IEEE.

Gracanin, D., Srinivasan, P. and Valavanis, K., 1993, May. Fundamentals of parameterized petri nets. In *[1993] Proceedings IEEE International Conference on Robotics and Automation* (pp. 584-591). IEEE.

Hamadi, R. and Benatallah, B., 2003, January. A Petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference-Volume 17* (pp. 191-200). Australian Computer Society, Inc..

Hinz, S., Schmidt, K. and Stahl, C., 2005, September. Transforming BPEL to Petri nets. In *International conference on business process management* (pp. 220-235). Springer, Berlin, Heidelberg.

Iordache, R. and Moldoveanu, F., 2014. QoS-aware web service semantic selection based on preferences. *Procedia Engineering*, *69*, pp.1152-1161.

Kristensen, L. M., Christensen, S. and Jensen, K., 1998. The practitioner's guide to coloured Petri nets. *International Journal on Software Tools for Technology Transfer (STTT)*, *2*(2), pp.98-132.

Laskey, K. B. and Laskey, K., 2009. Service oriented architecture. *Wiley Interdisciplinary Reviews: Computational Statistics*, *1*(1), pp.101-105.

Li, B., Sun, X., Leung, H. and Zhang, S., 2013. A survey of code□based change impact analysis techniques. *Software Testing, Verification and Reliability*, *23*(8), pp.613-646.

Llorens, M. and Oliver, J., 2004. Structural and dynamic changes in concurrent systems: reconfigurable Petri nets. *IEEE Transactions on Computers*, *53*(9), pp.1147-1158.

Murata, T., 1989. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), pp.541-580.

Newcomer, E. and Lomow, G., 2005. *Understanding SOA with Web services*. Addison-Wesley.

Novotny, P., Wolf, A. L. and Ko, B. J., 2013, May. Discovering service dependencies in mobile ad hoc networks. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)* (pp. 527-533). IEEE.

Petri, C. A., 1962. Kommunikation mit automaten schriften des rheinisch. Westfalischen Bonn. Translation by CF Green, Applied Data Research Inc., Suppl, 1., Bonn: Inst. fur Intrumentelle Mathematik and der Universitat.

Ross-Talbot, S. and Fletcher, T., 2006. Web services choreography description language: Primer. *World Wide Web Consortium, Working Draft*.

Van der Aalst, W.M., 1998. The application of Petri nets to workflow management. *Journal of circuits, systems, and computers*, *8*(01), pp.21-66.

Verbeek, H. M. and van der Aalst, W.M., 2005, June. Analyzing BPEL processes using Petri nets. In *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management* (pp. 59-78).

Wang, Y., Yang, J., Zhao, W. and Su, J., 2012. Change impact analysis in service-based business processes. *Service Oriented Computing and Applications*, *6*(2), pp.131-149.

Zhang, H., Li, J., Zhu, L., Jeffery, R., Liu, Y., Wang, Q. and Li, M., 2014. Investigating dependencies in software requirements for change propagation analysis. *Information and Software Technology*, *56*(1), pp.40-53.

Zhang, L. J., 2006, September. SOA and Web services. In *2006 IEEE International Conference on Services Computing (SCC'06)* (pp. xxxvi-xxxvi). IEEE.

Zimmermann, T., Zeller, A., Weissgerber, P. and Diehl, S., 2005. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, *31*(6), pp.429-445.