International Journal of Organizational and Collective Intelligence
Volume 13 • Issue 1

Comprehensive Framework-Based Reconfigurable Object Nets for Managing Dynamic Protocols Evolution

Radja Hamli, MISC Laboratory, University Constantine 2 – Abdelhamid Mehri , Constantine, Algeria
Allaoua Chaoui, University Constantine 2, Algeria
Raida Elmansouri, University Constantine 2, Algeria
Ali Khebizi, LabSTIC Laboratory, 8 May 1945 University, Guelma, Algeria
https://orcid.org/0000-0002-5372-8201

ABSTRACT

In the change management context, handling web service evolution is a challenging issue that aims to adapt deployed business processes to the perpetual changes occurring in enterprises environments. While existing approaches deal with the problem by focusing only on migratable instances, in this approach the authors propose a paradigm shift based on reconfigurable objects nets (RONs) to allow running service instances continuing their execution according to the evolved protocol specifications. In this approach, service protocols are modelled as petri nets, and changes are perceived as transformation rules. Further, reachability graphs are deployed for calculating migratable services instances after evolution. The conceived framework allows migrating active instances from the old protocol version to the evolved one. Web service protocol compatibility and replaceability aspects are addressed to distinguish migratable services instances from non-migratable ones. The authors illustrate their contribution and highlight advantages of using RONs through a real-world scenario related to the visa application service.

KEYWORDS

Business Protocols, Dynamic Protocol Evolution, Instances Migration, Reconfigurable Object Nets, Reconfigurable Petri Nets, Service Instances, Web Services

INTRODUCTION AND PROBLEM STATEMENT

Business processes constitute the cornerstone of modern organizations. Thus, companies invest huge efforts and sums for managing the associated life cycles while modeling and maintaining their processes schemes. Meanwhile, as a revolutionary technique in the software industry, Web services are becoming the new generation of software components allowing the implementation of various kinds of business processes. In fact, Web services technology is suitable for supporting the integration of distributed and heterogeneous information systems.

DOI: 10.4018/ijoci.318446 *Corresponding Author

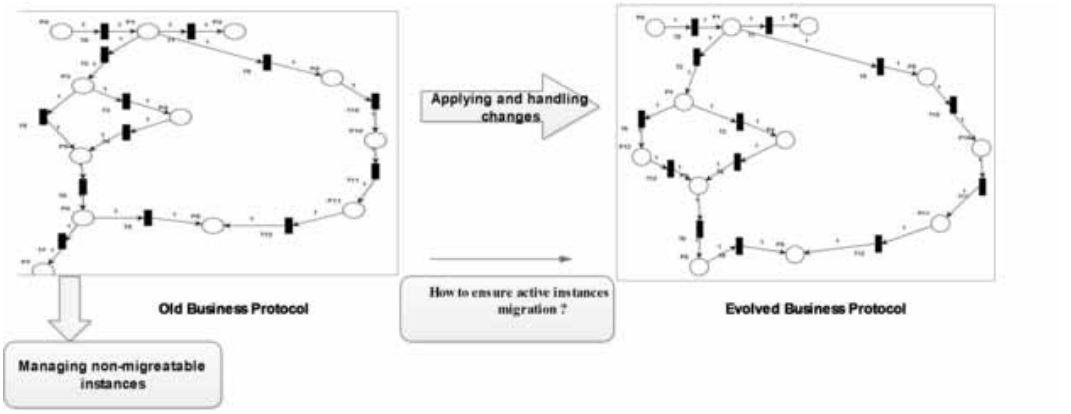
Two elements are fundamental to enhancing the interactivity level between the different stakeholders involved in Web services environments. First, the *service interface* describes static elements useful for invoking the available operations, like messages, types, port types, and the required protocol. However, as service operations cannot be invoked in an aleatory fashion, the *service protocol* represents an abstract tool to specify the operations sequences imposed by the service provider according to his business logic. In a nutshell, a service protocol is an abstract model which handles the sequence of messages that a service and its clients exchange to achieve a certain business goal (Alonso et al., 2004; Ryu et al., 2008; Benatallah et al., 2006). In this perspective, the research literature has highlighted the usefulness of service protocols, and various models performing different expressivity and relevance levels have been suggested to handle different types of constraints.

Nowadays, contemporary enterprises evolve in a turbulent environment from which they recover information related to customer needs, partners, and competitors, and for which they produce consumer goods for clients and other companies. Indeed, due to the phenomenon of economic globalization and as a consequence of advances in information and communication technology (ICT), the market has become global, and competition is increasingly hard. Consequently, enterprises must face a high level of dynamicity where evolutions become intrinsic aspects of such competitive environments. The development of modern corporations leads to their transformation into open systems that have close connections with a highly unexpected and unstable environment; in return, their survival is dependent on them. Many reasons, such as changes in the business logic or business strategies, the evolution of laws and regulations, and adaption needs, can lead to changes in the already deployed Web services specifications. Business protocol evolution involves updating an old protocol description, e.g., adding or removing activities or steps in the current procedure to comply with new business requirements (Khebizi et al., 2017). In this context, a crucial issue lies in the management of active instances having started their interactions based on the old service protocol version. In fact, stopping a system while execution instances are still running may involve a loss of historical work that has been accomplished. Therefore, the ability to ensure the system's continuation without stopping it should be an option. However, the problem is the management of the ongoing instances having started according to the previous protocol when it has been modified. To address such issues, a thorough analysis of active instances that need to comply with the business modifications is required. Hence, defining a seamless migration strategy must focus on two complementary aspects: the business protocol specification and the progression level of instances to be migrated to the new protocol (Khebizi et al., 2017). Ryu et al. (2008) proposed three migration strategies: continuation, migration to the new protocol, and migration to ad hoc protocol. Figure 1 illustrates the scenario of business protocol evolution and its challenging issues.

To face the challenge of managing dynamic business protocol evolution, high-level modeling techniques and formal methods are very useful for establishing a solid theoretical foundation that will facilitate the management of the problem in an adequate and formal fashion. Thus, the main goal of this work consists of focusing on abstract Web service specifications by using formal and abstract tools in order to address the problem of dynamic Web services evolution and to examine entailed impacts of change management. Among the panoply of abstract models that have proven their strength, soundness, and validity in various research areas, Petri nets are the most widespread ones in the field of dynamic and real time-systems. The Petri net formalism is intensively used to model, analyze, simulate, control, and evaluate the behavior of distributed and concurrent systems (Murata, 1989). Nonetheless, it is observed that basic Petri nets do not provide a direct method for handling difficulties induced by dynamic changes in systems. To overcome this limitation, this formalism was enhanced with enriched extensions allowing a formalization of different features.

In fact, the extension of basic Petri nets to reconfigurable Petri nets (*RONs* for short) was inspired by the evolution in software and hardware systems advances in the trend to manage dynamic systems changes. This improvement replaces the structures of classical systems from rigorous to flexible, open, and dynamic ones (Padberg & Kahloul, 2018). Reconfigurable Petri nets provide a soundness dimension to classical Petri nets that make the discrimination between the levels of change possible

Figure 1.
The challenge of managing dynamic business protocol evolution



due to the integration of a set of rules that can change the Petri nets (Padberg & Kahloul, 2018). Consequently, we are convinced that such formalism is a natural, effective, and suitable tool for handling dynamic business protocol evolution.

Dynamic change management of Web service business protocols is a major issue to be tackled in order to strive for rapid reconfiguration of business processes to adapt to a new and rapidly changing environment. Hence, in this work, we aim to ensure Web service instances execution continuation according to the new protocol specifications imposed by changes occurring in enterprise environments. To this end, we develop a formal technique that, on the one hand, enables the modeling of evolving processes as RONs, and on the other hand, supports changes by deploying transformation rules reflecting new specifications of processes. The main idea behind using RONs is the integration of transition firing and rule-based net structure transformation of place–transition (PT) nets during the evolution of Web services protocols. This can be achieved by the appropriate integration of token-nets and token-rules in a high-level net model. The conceived framework allows migrating a large spectre of active instances of the current service protocol. In fact, instead of considering running instances in a systematic manner as migratable and not migratable ones, we force them to adapt to the new requirements dictated by environment changes. The main benefits of this perception consist of guaranteeing a maximum rate of migratable instances and avoiding loss of work induced by the recovery from scratch of non-compatible instances.

The suggested method represents a new mechanism for tackling the issue of instance continuation in the business protocol evolution context. We propose a paradigm shift based on reconfigurable object nets (RONs). Based on such formal and sound tools, the conceived framework offers to protocol managers the possibility to continue the execution of ongoing instances according to the evolved protocol specifications. Whilst existing approaches for dynamic protocol changes impose active instances to be compatible as accurately as possible with the initial service protocol, our approach allows all active instances to be migrated by deploying the reachability graph that allows service providers the ability to manage the constraints that drive the instances migration process. Our contribution covers the following aspects:

- A formal verification of compliance criteria in order to ensure that instances migration does not suffer from problems such as deadlocks, activities re-executions, or deletion of already started activities.
- A maximum coverage of instances to be migrated by exploiting the reachability graph.
- An operational and actionable framework for managing instances migration that exceeds unsuccessful abstract and theoretical approaches.

Subsequently, the proposed approach adds value to the area of business protocol evolution. In fact, when building and deploying the reachability graph of a Petri net model, properties of the modeled system are performed through the marking paths from the initial marking to the target one. In this perspective, the constructed reachability graph is exploited to calculate compatible conversations that can be migrated to the new business protocol. Notably, in most of the existing approaches in the literature, services' active instances are managed and supported by users' predefined migration strategies (e.g., black and white), taking into account neither specific user needs nor complex migration strategies. To illustrate the applicability and feasibility of our approach, various RONs are manipulated in the paper, and the usage of the most suitable one for a specific evolution context is illustrated with real examples. Our suggestion for managing changes and handling protocols evolution operates in an incremental fashion.

First, we consider old and new service protocols as graphs, and we use graph transformation techniques and tools to model these protocols as Petri nets. Then, we make various changes to the initial service protocol (e.g., adding or removing activities by operating the adequate changes to the initial service protocol specification). At this stage, the formal model of RONs plays a crucial role, and the RON-Editor is deployed to simulate the business protocol evolution. This goal is reached by considering two dimensions of RONs, the system level and the token level, and their associated types of tokens: token-nets and token-rules. In the formal RONs model, token-nets are PT nets, while token-rules are double pushout production rules.

Once changes are performed and modeled with RONs elements (token-nets and token-rules), compatibility properties are taken into account as a discriminator factor to handle instances migration. Hence, the concept of the reachability graph of the RONs is exploited to calculate the active instances able to migrate to the evolved business protocol. Accordingly, to the work of (Ryu et al., 2007), in our approach, we classify active instances into two kinds, i.e., migratable and non-migratable ones, and we handle non-migratable instances by applying ad hoc protocols techniques. To achieve this goal, double-pushout rules are used to realize a protocol adapter that handles the mismatches and bridges the gap between old and new protocol versions. A real-world scenario (i.e., the Australian working visa application service) of a Web service business protocol is used throughout this paper to illustrate the proposed approach and to highlight the different facets supporting the formal concepts.

The remainder of this paper is organized as follows. After a literature review of the change management and evolution problem, we present materials and methods used in this work. The following section is dedicated to the results of managing changes by deploying RONs, and we present an instances classification technique (migratable, non-migratable) and study the problem of the non-migratable instances; this is followed by a discussion of our results. Finally, the conclusion and potential perspectives of our work are drawn.

LITERATURE REVIEW

The problem of evolution management has been tackled from different aspects in a panoply of research works. In fact, there have been various proposals regarding systematic approaches in many research areas on data and software engineering to ensure more flexibility to information system, as well as managing dynamic changes. In the business processes field, impacts of evolution have been approached from perspectives such as the improvement of business process reconfiguration and process instances migration. The application of changes to a certain service protocol, which is recognized as the dynamic evolution problem in Web services environments, was addressed based on three levels: the interface level, the business protocol level, and the instances level. The last two are important in dynamic service evolution and instance migration management. Hence, in what follows, we mainly focus on these aspects.

On the one hand, an important number of research works highlighted the subject of Web services business protocols evolution (Azough et al., 2009; Benatallah et al., 2005; Liske et al., 2009;

Papazoglou, 2008; Ryu et al., 2007, 2008; Skogsrud et al., 2007; Y. Wang & Y. Wang, 2013). On the other hand, another attempt to investigate the instance migration during protocol evolution by Y. (Wang & Y. Wang, 2013) implemented a business protocol model based on a finite state machine. In comparison, the automatic identification of migratable instances is achieved by the implementation of model-based techniques. This approach identifies the instances that are unaffected by the changes and easily migratable to the new protocol, while keeping the transparency of their migration to the evolved version. The main focus here is the deployment of the compatibility and replaceability (Benatallah et al., 2006) that are inevitable to safely identify migratable instances during the evolution of the protocol scheme. These notions are used to conduct a close analysis of protocol evolutions impacts.

By using forward and backward compatibility principles, various compatibilities classes were distinguished by Azough et al. (2009) and Liske et al. (2009) to specify the kinds of service instances migration. Comparatively, the asynchronous and non-deterministic service protocols evolution (Liske et al., 2009) extends the set of techniques used by Ryu et al. (2008); while Papazoglou (2008) introduced shallow changes that are confined to either services or clients. However, deep changes are changes where cascading effects and side effects take place.

In case a proper migration of active instances is not possible, adaptation techniques (Benatallah et al., 2005; Kaminski et al., 2006; Yellin & Strom, 1997) could be deployed to ensure execution continuation. In counterpart, (Benatallah et al., 2005) provides a technique to calculate protocol mismatches and similarities in order to exploit the computed differences for generating adequate adapters between different protocol versions. The computed adapters allow shifting old instances from the old protocol version to the new one. Weber et al. (2008) suggested a set of adaptation patterns to allow users structurally change process specifications and consequently facilitate version control for business processes' schema evolution. In opposition, Zhao and Liu (2013) provided a comprehensive method for navigating process instances executions of changing process versions.

To manage change impact, Dam and Ghose (2015) proposed an approach for analyzing the business protocol of a Web service by mining a version history of a business process model repository. On the other side, by Mafazi et al. (2014), suggested another approach to handle changes that are concurrently done by various stakeholders. The suggested approach provides solutions for the on-the-fly conflict changes through the implementation of behavior consistency rules of business processes. Similarly, Fdhila et al. (2015) proposed a generic change propagation approach focused on refined process structures in order to ensure propagating changes in a decentralized manner in a process choreography scenario.

Petri nets are mostly used in the modeling field, and a wide range of works implements this abstract formalism as a tool for managing change impact analysis and for handling issues occurring during systems evolution. Moreover, new extensions of this formalism were introduced to distinguish concurrent and dynamic systems by suggesting various enhancements of the basic Petri net model (Llorens & Oliver, 2004b; Badouel et al., 2003). In this context, Padberg and Kahloul (2018) classified reconfigurable Petri nets into three types: reconfigurable low-level nets, reconfigurable stochastic nets, and reconfigurable high-level nets. In addition, it is argued in the literature that various reconfigurable Petri nets approaches can be classified into three principal classes: rewriting net systems-based approaches, graph transformation-based approaches, and hybrid approaches.

Furthermore, changes and crises lead to rapid transformations to remote working and learning modes and the need for e-commerce, education-related project development, and maintenance. Moreover, an increase in internet traffic has a direct impact on infrastructure and software performance. Fedushko et al. (2020) studied the problem of accurate and quick Web-project infrastructure issues, bottleneck, and overload identification and explored methods for strategic management of Web projects. Chang et al. (2019) suggested a reuse strategic decision pattern framework (RSDPF) based on blending ANP and TOPSIS techniques, enabled by the OSM model with data analytics. J. Wang et al. (2020) introduced three approaches to predict online conformance through the construction

of a classification model automatically based on the historical event log and the existing reference process model.

As mentioned previously, an analysis of the state of the art covers a remarkable variety of methods for managing change impacts induced by evolution needs. However, the efficiency and performance of each proposed method will depend on the formalism used for specifying the protocol model and execution traces. Therefore, a robust mechanism to model the migration of active instances after protocols' evolution is inevitable. Furthermore, the success of any approach will depend on the formalism used. Before the presentation of our approach, we introduce in the next section the necessary materials and methods to make this paper self-contained.

MATERIALS AND METHODS

Materials

This section introduces the preliminary basic notions and definitions, allowing a clear comprehension of the paper, followed by a presentation of RONs and the motivations having led to using them in the context of changes and evolution of dynamic systems. Below, we present the concepts of PT nets, morphisms over PT nets, union, and transformations (Hoffmann et al., 2005; Kahloul et al., 2014).

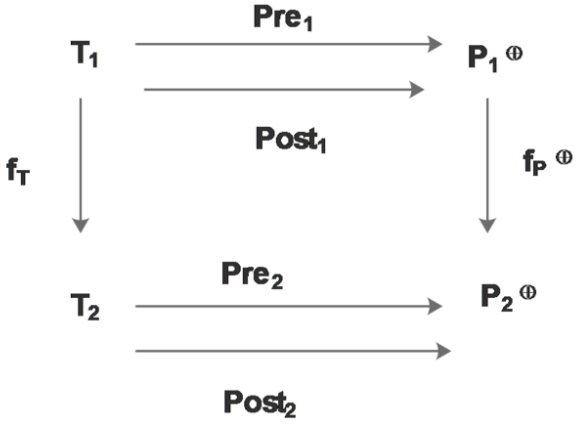
Definition: PT Nets

A PT net is formally represented by a quadruplet $\left(P, T, \Pr e, Post\right)$, where T is a non-empty finite set of transitions, P is a non-empty finite set of places, $\Pr e$ (for pre-domain) and $Post$ (for post-domain) are the two mappings defined as $\Pr e, Post : T \to P^{\oplus}$. The set $P^{\oplus}$ denotes the set of finite multi-sets over the set P (Hoffmann et al., 2005; Kahloul et al., 2014)

Definition: Morphisms Over PT Nets

A morphism between the two PT nets $N_1 = \left(T_1, P_1, \Pr e_1, Post_1\right)$ and $N_2 = \left(T_2, P_2, \Pr e_2, Post_2\right)$ is a function $f : N_1 \to N_2$. We have $f = \left(f_T, f_P\right)$, such that: $f_r : T_1 \to T_2$ and $f_P : P_1 \to P_2$ are two morphisms that map transitions into transitions and places into places, respectively (figure 2). The expressions f_T and f_P satisfy: $\Pr e_2 \circ f_T = f_P^{\oplus} \circ \Pr e_1$ and $\mathrm{P}ost_2 \circ f_T = f_P^{\oplus} \circ \mathrm{P}ost_1$.

Figure 2.
Morphisms on PT nets (Hoffmann et al., 2005; Kahloul et al., 2014)



Definition: Union of PT Nets (Pushout)

The union is a specific construction based on the morphisms defined over PT nets $N_1 = \left(T_1, P_1, \Pr e_1, Post_1\right)$, $N_2 = \left(T_2, P_2, \Pr e_2, Post_2\right)$ and $I = \left(T_0, P_0, \Pr e_0, Post_0\right)$ with the two morphisms $f : I \to N_1$ and $g : I \to N_2$. The net I denotes a common interface between N_1 and N_2. The union of N_1 and N_2 is the Net $N = \left(T, P, \Pr e, Post\right)$, defined using the two morphisms $f' : N_1 \to N$ and $g' : N_2 \to N$. We write $N : N_1 +_I N_2$. The operator $+_I$ is called the pushout construction or the gluing operator.

Definition: Transformations of PT Nets (Double Pushout)

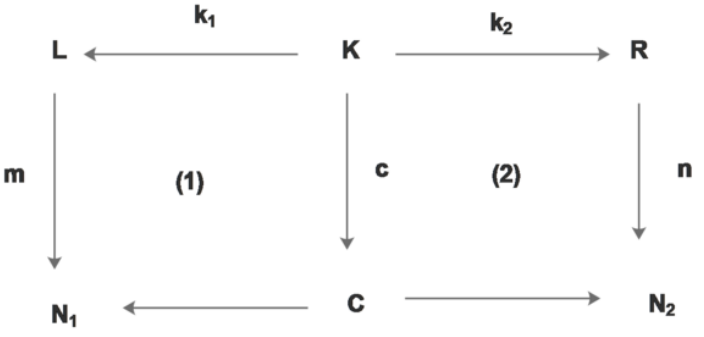
The transformations of PT nets, also called *double pushout*, are based on the PT gluing (or pushout) construction. Let L, K, R and C be four PT nets. A transformation $f : N_1 \to N_2$ transforms the PT net N_1 into PT net N_2 using the rule $r = \left(L, K, R\right)$ and the match $m : L \to N_1$ iff we have the double pushout in Figure 3.

In Figure 3, $k_1, k_2, m, c,$ and n are morphisms. Thus, the PT net C is called the context of the transformation and it satisfies the following conditions:

1. $T_C = \left(T_1 \setminus m_T\left(T_L\right)\right) \cup m_T\left(k_{1_T}\left(T_K\right)\right)$;
2. $P_C = \left(P_1 \setminus m_P\left(P_L\right)\right) \cup m_P\left(k_{1_P}\left(P_K\right)\right)$;
3. $\Pr e_c = \Pr e_{1|T_C}$ (The relation $\Pr e_C$ is the subset of $\Pr e_1$ which concerns only the set of transitions: T_C).
4. $Post_C = Post_{1|T_C}$ (The relation $Post_C$ is the subset of $Post_1$ which concerns only the set of transitions: T_C).

Furthermore, the incessant need to model and simulate dynamic systems, as well as to specify and manage their changing properties, has been felt in different research areas (Padberg & Kahloul, 2018). To this end, the notion of reconfiguring Petri nets was introduced in the early nineties (Ehrig et al., 1994), and it underwent levels of formalization by many researchers (Padberg & Kahloul, 2018). These abstract tools can be considered as a family of formal modeling techniques allowing a

Figure 3.
The double pushout mechanism (Hoffmann et al., 2005)



variety of Petri net types, such as: high-level nets, timed or stochastic nets, and object nets (Padberg & Kahloul, 2018). Reconfigurable Petri nets consist of marked Petri nets, i.e., a net with a marking and a set of rules that modifies the net's structure at run-time when it is applied (Ehrig et al., 2007; Ehrig & Padberg, 2004; Llorens & Oliver, 2004a; Prange et al., 2008). As a witness of the richness and the relevance of the usefulness of RONs, they were deployed in various research and industry areas (Padberg & Kahloul, 2018) and particularly in reconfigurable manufacturing systems (RMSes; Kahloul et al., 2016), flexible manufacturing systems (Tarnauca et al., 2012), workflows in dynamic infrastructures (Hoffmann et al., 2008), and concurrent systems (Llorens & Oliver, 2004b).

This interest in using RONs is motivated by their high level of expressiveness and ability to specify constraints inherent to dynamic systems. From a specification perspective, RONs were initially introduced as high-level nets with nets and rules as tokens (Biermann & Modica, 2008; Hoffmann et al., 2005). The main constructions of graph transformations used in Petri nets are the union (single pushout) and transformation (double pushout). The formulation of these operations uses a set of morphisms over nets. In the following, we recall some definitions of RONs, and we highlight their deployment for managing changes and transformations (Ehrig & Padberg, 2004; Kahloul et al., 2014). To understand the various criteria of RONs, it is essential to mention that these formalisms are based on graph transformation theory (Ehrig & Padberg, 2004). A RON model is a complex structure, having, on the one hand, the token level, on which nets represent token-nets and double pushout production rules express token-rules. On the other hand, it also includes the system level in which we can distinguish two types of places depending on the system specification: net-places or rule-places, i.e., it includes token-nets or token-rules, respectively (Kahloul et al., 2014). Nevertheless, when transitions in the system level trigger the dynamic behavior over markings of the token-net, they are considered as fire-transitions, which are expressed as follows.

A transition t from a net, in which t updates the marking of N by firing t when the latter is enabled. A fire-transition requires a guard $\left[enabled\left(t\right) = true\right]$ (Kahloul et al., 2014). A new net computed using the function: $fire\left(N;t\right)$ is produced by this fire-transition once it is fired (Kahloul et al., 2014). However, when transitions trigger the reconfiguration behavior over the structure of the token-nets through the application of token-rules, they describe transform-transitions as having the following parameters.

Let $R = \left(p;m\right)$ be a rule used for transforming a net, with p a PT net and m a morphism. To be activated, the transform-transition mechanism supported by the rule R requires an applicability constraint named the guard [$applicable\left(N;R\right)$] (Kahloul et al., 2014). Hence, the resulting net having a new structure defined through a function: $transform\left(N;R\right)$ is produced as the output of the considered transform-transition function (Kahloul et al., 2014).

Given their semantic richness and their theoretical foundation, RONs have benefited from a particular interest, both from the research community and from software firms. Nowadays, it is observed that various providers offer free and downloadable software tools such as RON-Editor (Biermann et al., 2007) and ReConnect (Ede et al., 2012) that are used to simulate and analyze RONs models. Further, such tools allow the representation of the system functionalities, where the dynamic at the micro-level and macro-level, the applied transformations' rules, and the system's configurations set are represented. Moreover, RONs grant the ability to use reconfigurable manufacturing systems (RMSes; Kahloul et al., 2014). Also, the application of graph transformation theory to a variety of Petri net kinds: Algebraic high-level nets, PT nets, and colored Petri nets are some of the RON's most prominent features (Kahloul et al., 2014).

Presentation of the RON-Editor

To show the applicability and feasibility of the proposed approach, we implemented and experimented with the RON-Editor (Biermann et al., 2007) tool. The RON-Editor (Biermann et al., 2007) is an open-source software tool that provides useful functionalities for managing the life cycle of Petri net

models. Furthermore, it allows handling evolution rules of specified Petri nets by taking into account occurring transformation rules. Thus, it offers protocol managers an efficient and easy environment that facilitates the description of system evolution.

In the RON-Editor tool, business protocols are perceived as Petri nets (PT systems), and their evolution is managed by specifying the set of transformation rules. Furthermore, the RON-Editor supports the consistency of the RONs by ensuring that rules' mappings fulfill the features of net morphism and that RON places have correct token typing (Padberg & Kahloul, 2018). To realize simulation scenarios, the RON-Editor contains an Attributed Graph Grammar engine (AGG; Graph Grammar Group, n.d.). Also, it uses Eclipse Modeling Framework (EMF) and Graphical Editor Framework (GEF) plug-ins developed as visual editors (Padberg & Kahloul, 2018). Reinforced with the previous components, using RON-Editor allows for a large spectrum of functionalities, e.g., object nets and net transformation rules, as well as top-level RONs. Therefore, it enables updating and managing the evolution of models as high-level transformations materialized by transitions to be fired within the models in the RONs editor (Padberg & Kahloul, 2018)

Methods

This section presents the different aspects of our approach intended to manage the dynamic evolution of business protocols and analyze change impact on active instances. In our approach, Web services business protocols are formalized as Petri net models. This choice is motivated by the dynamicity underlying the associated formal tools. Then, we deploy RONs for capturing changes as transformation rules, and we formalize protocol evolution by deploying the formalized rules. Once changes are taken into account, we describe the needed algorithms for classifying active instances into migratable and non-migratable ones, and we specify algorithms for ensuring the migration of active instances of Web services. More precisely, our approach for handling business protocol changes and ensuring active instances continuation is articulated around four complementary steps.

1. First, we present the case study that will be used throughout the paper to illustrate our approach, and we formalize it by using the Petri net formalism.
2. Based on the specified service protocol model, changes are handled by using RONs. Hence, protocol operations for handling changes are seen as double pushout rules, and the underlying modifications of the model are perceived as reconfigurations of the model. In this context, graph transformation techniques and tools are used to express changes occurring during the evolution of business protocols.
3. In the third step, we exploit the concept of the reachability graph to calculate the set of instances to be migrated to the new business protocol version. However, using RONs as dynamic structures leads to a problem during the reachability graph construction phase. Thus, a new reachability graph algorithm that allows for the analysis and verification of the RONs is proposed at this stage.
4. We analyze the impact of protocol changes based on backward and forward compatibilities properties. According to this analysis, we classify the ongoing instances into migratable and non-migratable ones. Lastly, handling the issue of non-migratable instances is tackled by using ad hoc protocols or adapter protocols. In this perspective, adequate algorithms are specified.

Presentation of the Case Study

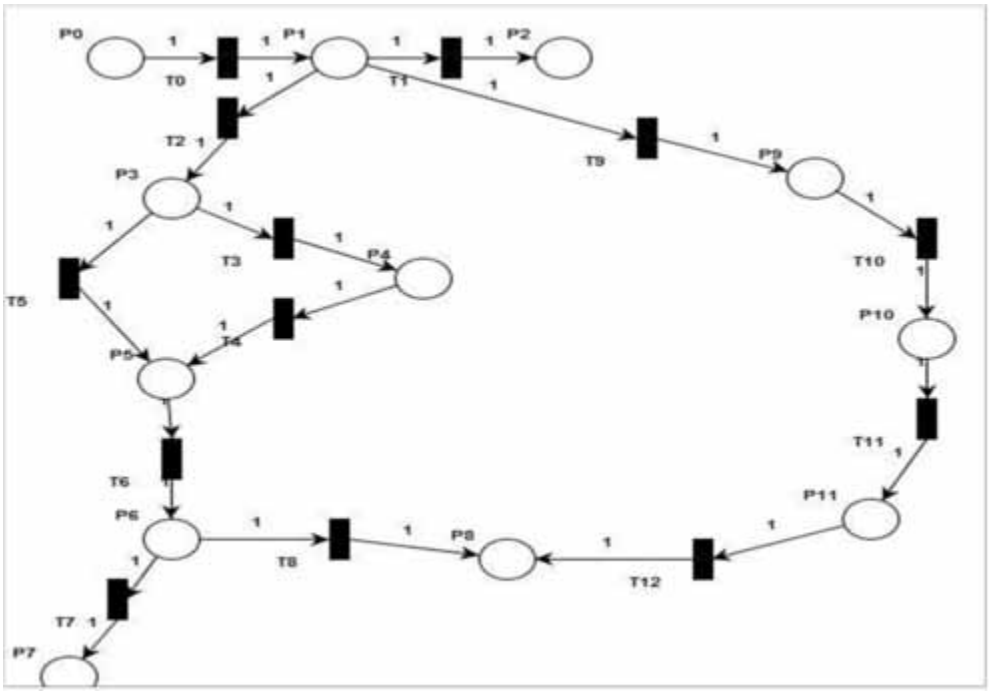
To illustrate the proposed approach, we start by exposing a real-world example of an Australian working visa application service inspired by the work of Ryu et al. (2008), which will be used through the remainder of the paper to highlight different facets of the proposed approach. This service protocol is exhibited in Figure 4, and the interpretation of the set of nodes in these token-nets is presented in Table 1.

The previous protocol can be deployed by the immigration department to manage working visa applications for immigrants. It is worth noting that at any given time, tens of thousands of protocol

Table 1.
Interpretation of nodes in the Token-nets protocol

Places		Transitions	
Symbol	**Interpretation**	**Symbol**	**Interpretation**
P0	Start	T0	Check eligibility
P1	Eligible	T1	Cancel
P2	Cancelled	T2	Fill in application
P3	Application ready	T3	Submit work experience
P4	Work experience submitted	T4	Test English ability
P5	Lodged	T5	Submit reference letter
P6	Checked	T6	Checked approval
P7 P8 P9 P10 P11	Reviewed Processed Student application ready Graduation certificate submitted Student logged	T7 T8 T9 T10 T11	ReassessConfirmFill in application for overseas student Submit graduation certificateSubmit passport
P12	Submit reference letter	T12	Check approval
		T13	Report medical examination

Figure 4.
Token-net ($TN1$) for an Australia working visa application service



conversations (instances) are in an active state and each of which has reached a particular progression level. However, many reasons can induce changes in the regulation governing the immigration procedure. Consequently, the amendment of immigration laws may affect the visa application protocol

(Ryu et al., 2008). As an illustration, the new laws can stipulate that (i) after the expiration of a working visa, if an applicant decides to reapply, he must submit an employer reference letter and the result of a medical examination in order to be accepted and (ii) reviewing the result of the application is no longer an option for the applicants. These changes cannot be delayed and must enter into force immediately. Obviously, the continuation of active instances having started their interactions based on the old protocol version is compromised, and a change impact analysis must be conducted to ensure their continuation in the realm of the evolved protocol version. To achieve this goal, first of all, the occurring changes must be handled and formalized in the new service protocol specification. The next subsection is dedicated to this aspect.

Handling Changes by Reconfiguring RONs

As RONs are used in our approach to representing service protocols, thus the occurring changes are perceived as a simple reconfiguration of these specifications acting as double pushout rules. To this end, two modeling levels are to be distinguished in order to implement RONs in a manner that can handle protocol evolution. First, the token-nets in the token level, defined as nets, allow for the exhibition of the static structure as well as the dynamic behavior of the considered business protocol. Further, the identification of the token-rules as double pushout rules conducts the reconfigurations of the specifications expressed with PT nets. Second, at the system level, places are either rule-places or net-places, depending on whether they include token-rules or token-nets, and transitions are fire-transitions and transform-transitions (Kahloul et al., 2016).

To highlight our propose, we operate the same modifications as proposed by Ryu et al. (2008).

1. Firstly, a place and a transition have been added: the new place *ReferenceLetterSubmitted* (P12) is inserted after the place *ApplicationReady* (P3), and the transition *ReportMedicalExamination* (T13) links the new place *ReferenceLetterSubmitted* (P12) to the *Lodged* one (P5).
2. The second change consists of removing both the *Reviewed* place (P7) and the associated transition *Reassess* (T7).

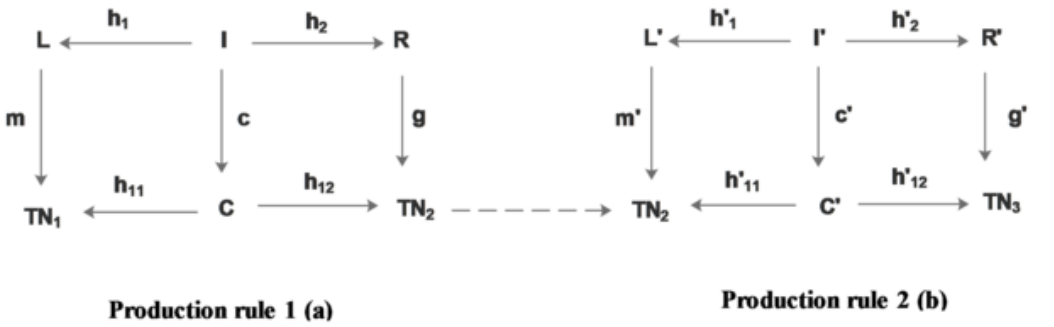
To update the service protocol in a fashion that meets the new specification of the visa application service protocol, an evolved version that integrates the occurring changes must be redesigned. In our context, changes in the Australian working visa application service require the definition of two production rules leading to a target service protocol which is a consequence of the business protocol reconfiguration. Moreover, a set of morphisms is necessary for constructing the two production rules.

In what follows, we focus on the modifications of the business protocol, and we specify the underlying modifications as double pushout rules. For the Australian working visa application protocol, changes to the initial service protocol are materialized with two production rules, as shown in Figure 5. As illustrated in the Figures, applying the pushout rules $R1$ and $R2$ ensures the transformation of the initial token-nets $TN1$ to an evolved one $TN2$, which will be in turn transformed by the second rule $R2$ to realize to the target token-net $TN3$.

Activation of the Double Pushout Rule $\boldsymbol{R}1$

The first double pushout rule is expressed with: $R1 = \left(p, m\right)$, where $p = \left(L, I, R\right)$ is composed of three P / T sub-nets (with L = left, I = Interface, and R = Right), and m is a morphism (see Figure 5A). More formally, the transformation associated with the rule $R1$ is expressed with $TN_1 \overset{(p;m)}{\rightarrow} TN_2$. This transformation involves the first reconfiguration of the service protocol from token-nets $TN1$ of Figure 4 to the target token-nets $TN2$ of Figure 9. Figure 6 showcases this rule, and it illustrates the double pushout mechanism which triggers the first reconfiguration of the token-nets $TN1$ to the token-nets $TN2$, according to the context C.

Figure 5.
An overview of the double pushout rules



As it appears in Figure 6, the place *ReferenceLetterSubmitted* (P12) and the transition *ReportMedicalExamination* (T13) are added to the right sub-net (R). The context C of Rule 1, as presented above, is depicted in Figure 7. For illustrative reasons, Figure 8 is a deep presentation of the underlying two morphisms h_1 and h_2 associated with the pushout rule $R1$. Once the double pushout construction of the rule $R1$ is applied to the initial protocol represented with the token-nets of Figure 4, a resulting service protocol depicted in Figure 9 is obtained. It is worth recalling that the resulting token-net of Figure 9 represents only an intermediate protocol version that handles changes expressed by the production rule $R1$. This token-net will serve as input for the second production rule $R2$. Hereafter, we focus on the second rule $R2$ and its specifications.

Activation of the Double Pushout Rule $R2$

The second double pushout rule $R2$ is formalized as follows.

$R2 = \left(p';m'\right)$ where $p' = \left(L',I',R'\right)$, consists to a three P/T sub-nets, (with L': left, I': Interface, and R': Right) and m is a morphism (see Figure 5B). The transformation associated with the rule $R2$ is expressed as $TN2 \xrightarrow{(p';m')} TN3$. In terms of formal specifications, the morphisms associated with the rule $R2$ are $h_1^{'}, h_2^{'}, h_{11}^{'}, h_{12}^{'}, c^{'}, \mathrm{m}', \text{and } \mathrm{g}'$, while $\mathrm{C}^{'}$ expresses the context of the rule (see Figure 5B). Figure 10 showcases the relationship between the different elements $\left(L',I',R'\right)$ managed by the double pushout rule 2. Once it is triggered, this rule allows the second reconfiguration of the protocol by taking the token-net $TN2$ of Figure 9 as input, which will be transformed into the target token-net $TN3$ of Figure 11. (The context $C^{'}$ of the rule $R2$ and the associated morphisms $h_1^{'}, h_2^{'}, h_{11}^{'}, h_{12}^{'}, c^{'}, \mathrm{m}', \text{and } \mathrm{g}'$ are not illustrated for lack of space).

The System Level Net

As it is argued previously, changes in the protocol specifications are governed by the RONs model of the system during the application of successive transformations. Two types of transitions in the system level are to be distinguished: fire-transitions and transform-transitions. While a transform-transition changes the structure of the token-net, the fire-transition one changes the marking of the token-net. Figure 12 shows the relationships between the various fire-transitions, transform-transitions, and the considered token-nets. As it appears in the figure, three net-places $\left(np1, np2, np3\right)$, containing respectively ($TN1$ in Figure 4, $\mathrm{T}N2$ in Figure 9, and $TN3$ in Figure 11) the previous token-nets form the system-kernel (*see the rounded forms at the right side of Figure 12*).

Figure 6.
Double pushout of the rule $\mathbf{R}1$

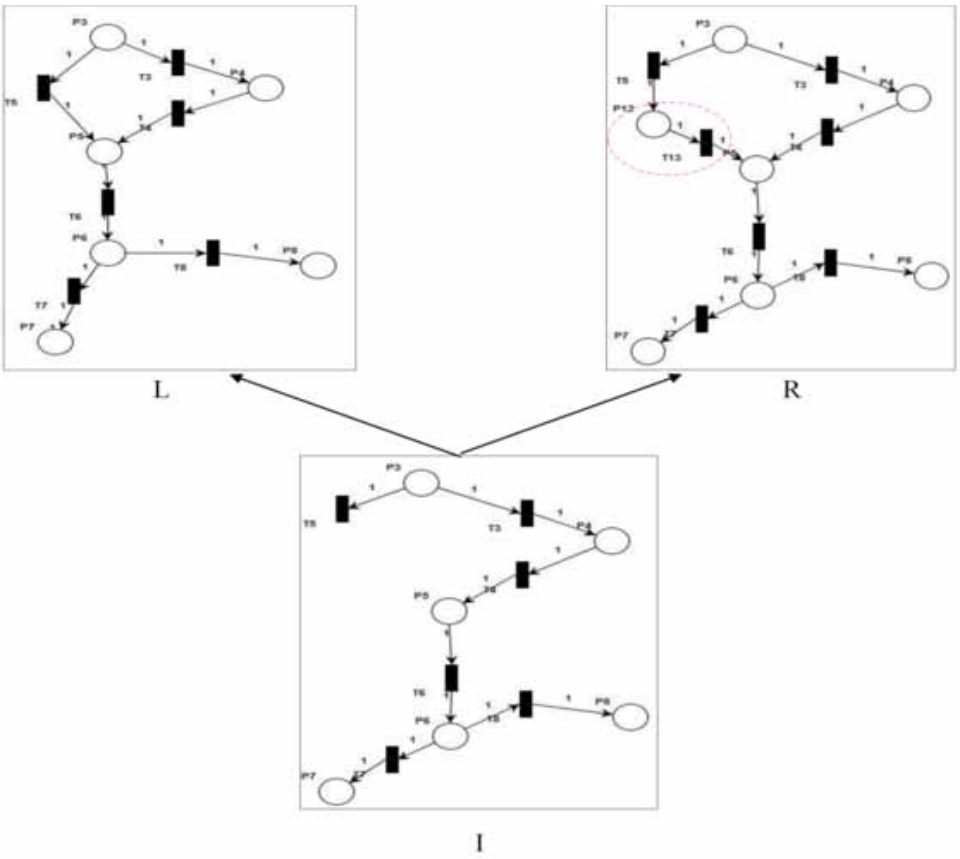


Figure 7.
The context $\mathbf{C}$ of the transformation based on the rule $\mathbf{R}1$

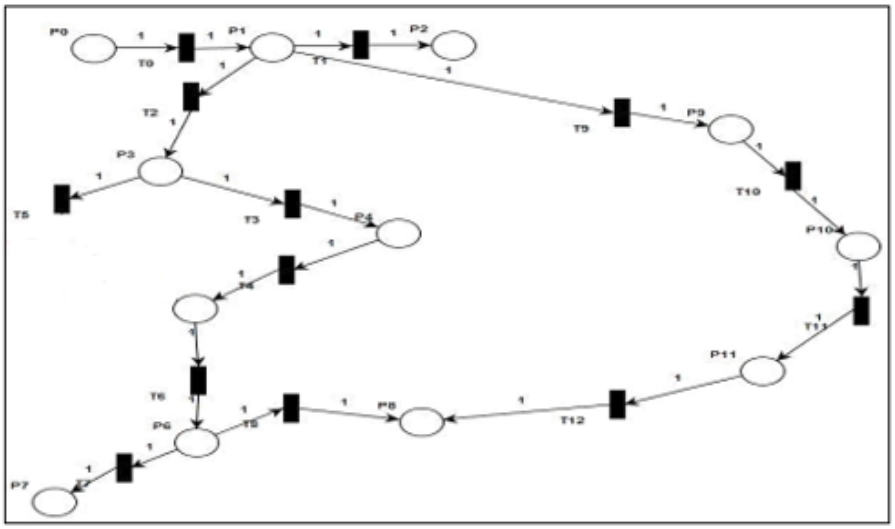
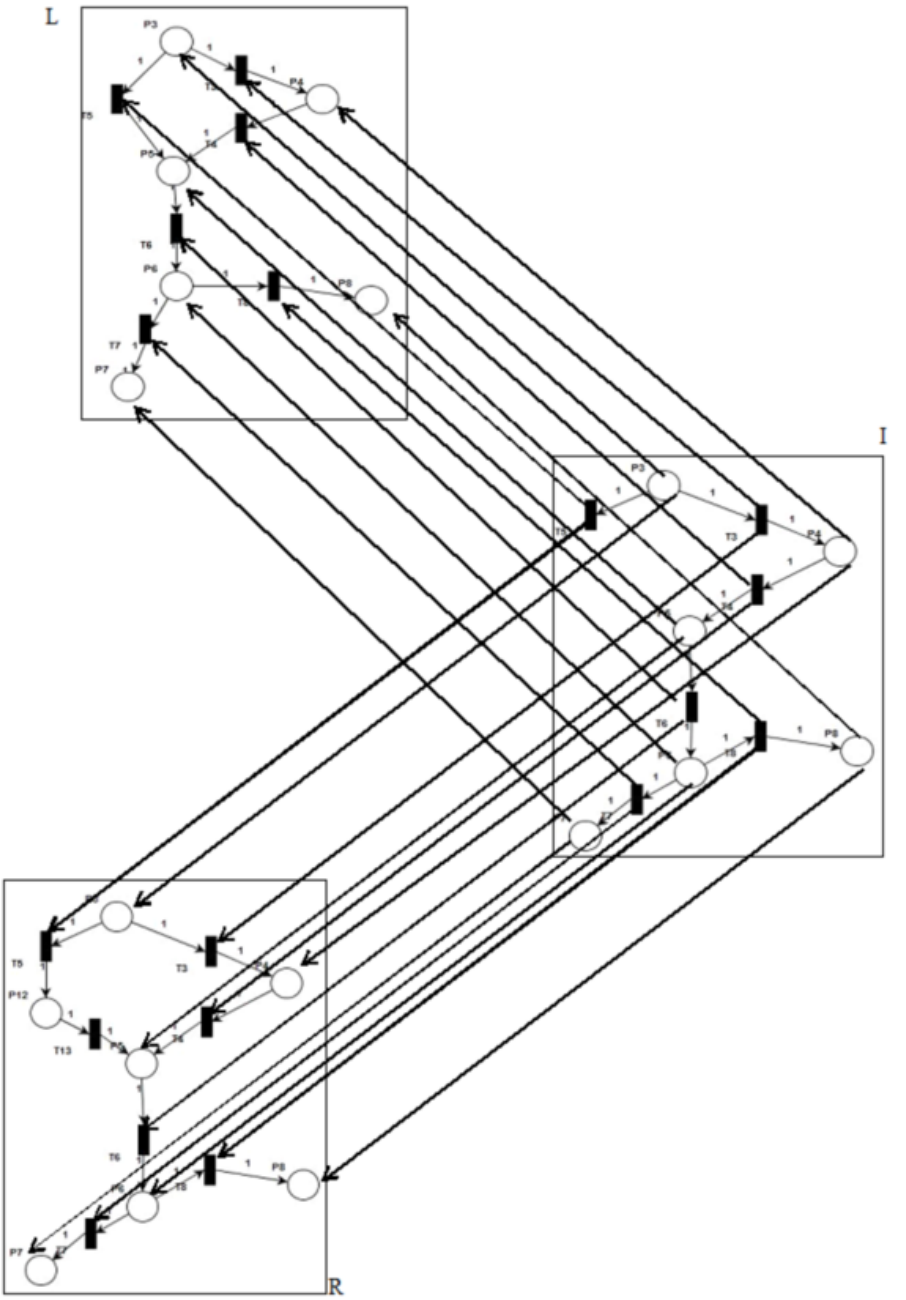


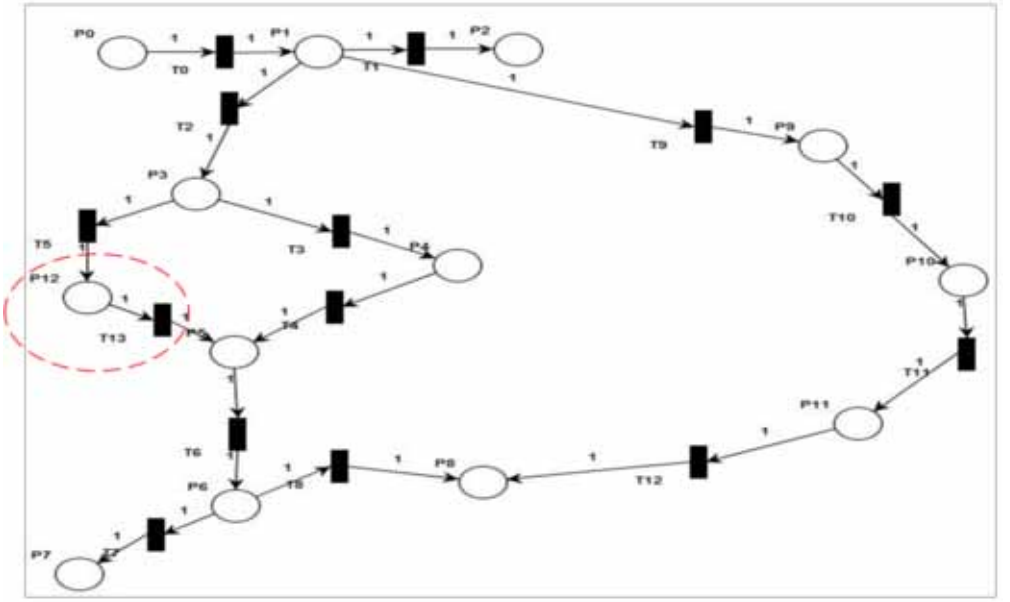
Figure 8.

The morphisms $\mathbf{h}_1$ and $\mathbf{h}_2$ underlying the first rule $\mathbf{R}1$



As shown in the top of Figure 12, the first configuration of the reconfigurable system occurs when the initial token-nets $TN1$ marks the net-place $np1$. Then, the markings of $TN2$ and $TN3$ are activated, respectively, through the application of the token-rules $\left(R1, R2\right)$ in Figure 5. Furthermore, it is observed in the Figure that two rule-places $\left(rp1, rp2\right)$, expressing, respectively,

Figure 9.
The resulting token-net $\mathbf{TN2}$ of the reconfiguration based on $\mathbf{R1}$



the two initial markings token-rules $\left(R1, R2\right)$, are highlighted. The figure illustrates that the final PT net $TN3$ is obtained by applying consecutive fire-transitions and transform-transitions, which are triggered according to particular guards.

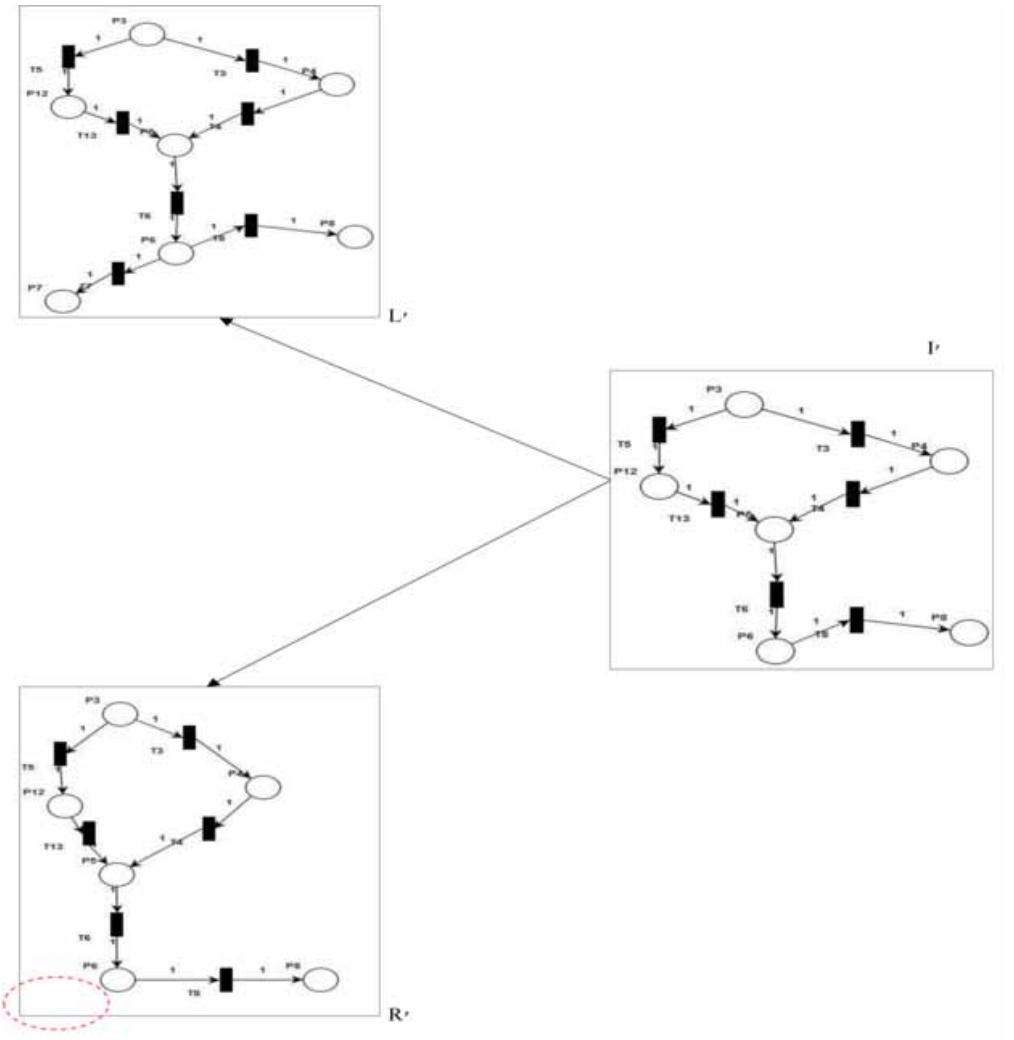
In our example of the working visa application service, both fire-transitions and transform-transitions, must have guards: $\left[\text{enabled}\left(t\right)\right]$ for fire-transition and a guard $\left[\text{applicable}\left(N;R\right)\right]$ for achieving transform-transition. Token-nets of the system (e.g., $TN1$) are linked to fire-transitions (e.g., fire-transition one) via specific guards of type *fire* (e.g., $\text{fire}\left(TN1,t\right)$). (See the top-left side of the figure). On the other hand, the transform-transition allows shifting token-nets of the system from an old configuration to an evolved one via transfom-transitions (e.g., transform-transition 1) under constraints of applicable guards (e.g., $\text{applicable}\left(TN1,R1\right)$), as depicted in Figure 12.

Managing Active Instance Migration

The third stage of our approach consists of managing active instances migration by exploiting the previously conceived models. We must now focus on the issue of ensuring active instances continuation. This is conducted by analyzing the ongoing instances. In this sense, both instances reached states, and their historical achieved activities are examined. This exploration allows splitting instances into two classes. Instances compatible with the operated changes are considered migratable, contrarily to those instances that cannot meet changes and are considered non-migratable. For populating these two instance classes with their suitable instances, the notions of backward and forward compatibility are used as a foundation and guidelines (Ryu et al., 2008).

In our approach, the identification and classification of active instances into migratable and non-migratable ones relies on the concept of a reachability graph for RONs. Such a graph allows deploying the principles of compatibility and replaceability to distinguish the instances that can be safely migrated when their protocol is modified. In this perspective, we operate in three incremental stages. First, we highlight the usefulness of constructing the reachability graph of the evolved system,

Figure 10.
Double pushout of the rule $\mathbf{R}2$

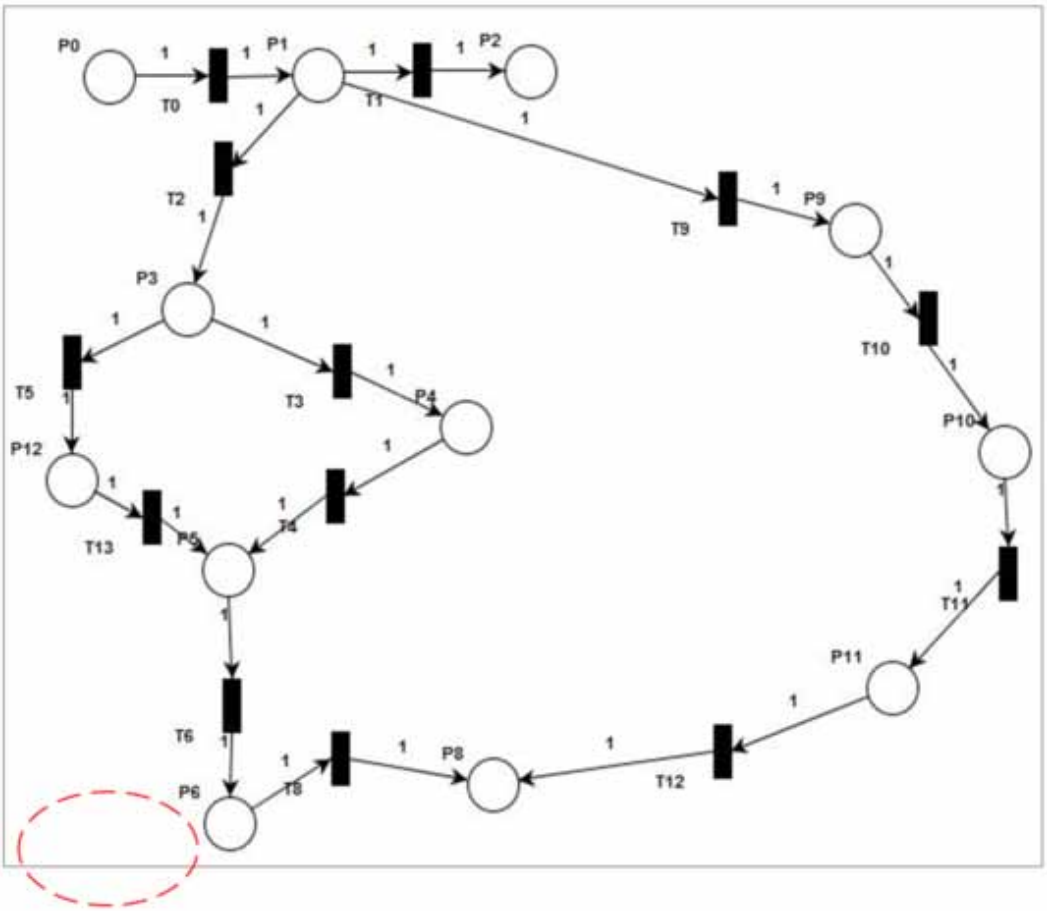


and then we elaborate on an algorithm for its generation. After that, we describe forward and backward compatibilities algorithms to ensure the migration of active instances to the new protocol version, according to the constructed reachability graph.

Generation of the RONs Reachability Graph

By using RONs for managing protocol evolution, the issue of managing active instances migration is relegated to calculating the reachability graph of the protocol and then exploring it for filtering migratable instances from non-migratable ones. Thereby, the reachability graph constitutes a "code of good behavior" and a guideline that active instances must comply in order to continue their execution in the realm of the new protocol. Consequently, instances that cannot meet the specification described in the reachability graph cannot continue their execution and, thus, are considered non-migratable ones. However, constructing such a graph is not an obvious task. In fact, in the context of ordinary

Figure 11.
The resulting token-net $\mathbf{TN3}$ of the reconfiguration based on $\mathbf{R2}$

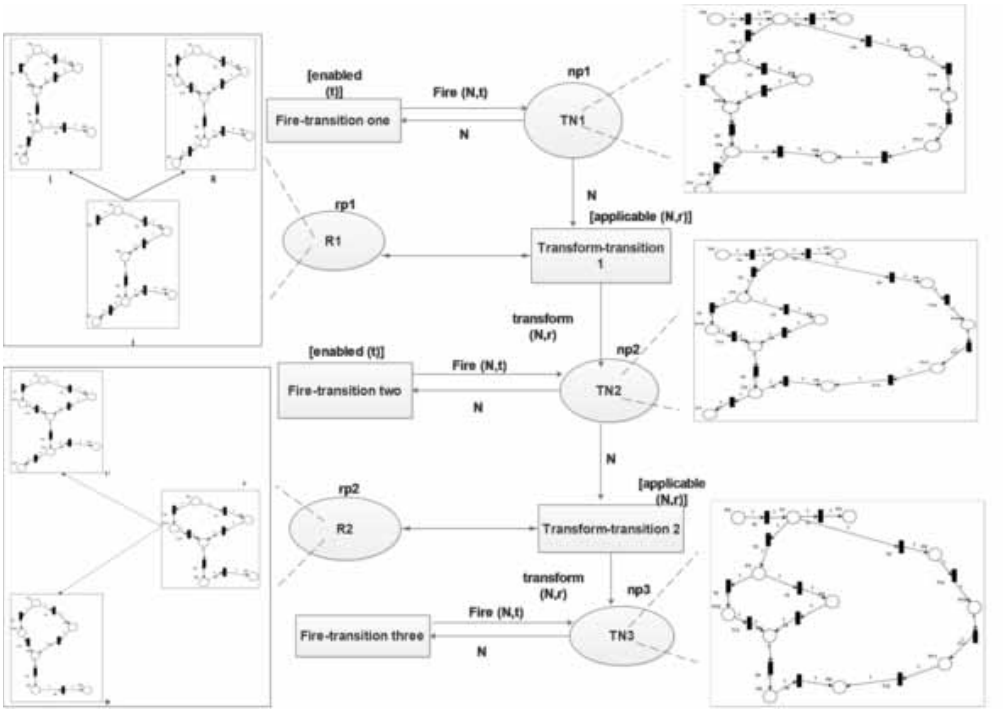


Petri Nets, reachability graphs are constructed without any particular difficulties, but the problem arises when dealing with formalisms having dynamic structures, such as RONs.

In what follows, we propose a detailed algorithm, named the reachability graph generator algorithm (RGGA) for generating the reachability graph based on an initial RON introduced in input and according to a set of transform-transitions (abbreviated as TT), as well as a set of fire-transitions (abbreviated as FT). In the algorithm, vertices represent places and their markings, while edges allow linking these vertices.

```
Algorithm 1: RGGA « The Reachability Graph Generator Algorithm »
Input: start
Output: vertices and edges of the reachability graph
begin
1.          vertices: = new ArrayList<>();
2.          vertices.add(start);
3.          edges = new HashMap<>();
4.          List <HighLevelPetri> q: = new ArrayList<>();
5.          q.add(start);
6.          HighLevelPetri curr: = null;
```

Figure 12.
The RON model of the system




```
7.          while (!q.isEmpty()) do
8.          curr: = q.remove(0);
9.          Set <String> firables: = curr.getFirable().keySet();
10.          if (!firables.isEmpty()) then
11.          List <String[]> el: = edges.get(curr.getName());
12.          if (el == null) then
13.          edges.put(curr.getName(), new ArrayList <> ());
14.          endif
15.          endif
16.          HighLevelPetri copy: = null;
17.          for (String hlt: firables) do
18.          copy: = curr.copy();
19.          HighLevelTrans t: = copy.getFirable().get(hlt);
20.          if (t instanceof FireHLTrans) then
21.          int fiSize: = ((FireHLTrans) t).getFi().size();
22.          for (int i = 0; i < fiSize; i++) do
23.          copy: = curr.copy();
24.          t: = copy.getFirable().get(hlt);
25.          List <Transition> fi: = ((FireHLTrans) t).getFi();
26.          for (int j = 0; j < i; j++) do
27.          fi.remove(0);
28.          endfor
29.          String n: = fi.get(0).getName();
30.          t.fire();
```

```
31.         HighLevelPetri id: = contains(copy);
32.         List <List<String>> path: = new ArrayList<>();
33.         curr.getPath().forEach(l->path.add(new
ArrayList<>(l)));
34.         path.forEach(l->l.add(n));
35.         if (id == null) then
36.         copy.setName("HLP"+vertices.size());
37.         copy.setPath(path);
38.         copy.setStartIndex(curr.getStartIndex());
39.         vertices.add(copy);
40.         q.add(copy);
41.         edges.get(curr.getName()).add(new String[]{/*t.
getName()+SEP+*/n, copy.getName()});
42.         else
43.         id.getPath().addAll(path);
44.         edges.get(curr.getName()).add(new String[]{/*t.
getName()+SEP+*/n, id.getName()});
45.         endif
46.         fi.remove(0);
47.         endfor
48.         else
49.         t.fire();
50.         HighLevelPetri id: = contains(copy);
51.         List <List<String>> path: = new ArrayList<>();
52.         curr.getPath().forEach(l->path.add(new
ArrayList<>(l)));
53.         int st: = Integer.parseInt(t.getName().charAt(2)+"");
54.         if (id == null) then
55.         copy.setName("HLP"+vertices.size());
56.         copy.setPath(path);
57.         copy.setStartIndex(st);
58.         vertices.add(copy);
59.         q.add(copy);
60.         else
61.         id.setStartIndex(st);
62.         if (path.size() > id.getPath().size())then
63.         id.setPath(path);
64.         endif
65.         endif
66.         endif
67.         endfor
68.         endwhile
end
```

Description of the Algorithm RGGA

All the graph components are RONs with different token-net positions and markings (lines 1–2). The initial RON *start* and having the name HLP0 is introduced as input of the algorithm RGGA. At first, the algorithm prepares the components (each vertex) of the graph. The algorithm uses a queue to explore the children of a node (lines 5–6), and it stores the vertices in a list whenever a new one is discovered, while edges are stocked in a map (lines 3–4). The key of this map is the name of a

vertex, while the value is a list of child vertices that are generated by applying a transition from this parent node.

The RGGA algorithm starts without any vertex in the queue, except for the initial RON one *start* (lines 3–4). During the algorithm progression, new vertices are added to and removed from the queue until it is empty. If this last condition is satisfied, the algorithm stops (the big loop from line 7 to the end). The algorithm takes the first element (RON) in the queue and removes it (line 8). This element is the variable *curr* in the algorithm. Now, all the fireable transitions of that RON are found (line 9) independently from their type (transform or fire transitions). This list of transitions is executed (fired) one by one (lines 17–30). After each transition firing, a new RON is obtained (line 30). At this stage, the parent RON *curr* and the new child RON *copy* are created. Therefore, two possible cases are to be distinguished, according to the fired transition type having generated a *copy*: a fire-transition (FT) or transform-transition (TT). In the first case (FT), the *copy* is tested to distinguish if it is a new vertex or if it already exists in the vertices set. In the case where it is a new one, it is added to the vertices set (lines 35–41) and to the queue structure (line 40). Furthermore, it is added the edge from *curr* to *copy* fired by this FT to the map of edges (line 41). However, if *copy* already exists in vertices (line 48), it is only added to the edge from *curr* to *copy,* which is fired by this FT to the map of edges (line 44). Then, all possible paths (fired transition sequence) that lead to *curr* from the root vertex *start* (line 43) are checked, concatenated in the new transition for each path (line 44), and stored in *copy*. Now, if the transition is transform-transition (TT), here again, we examine if *copy* is a new vertex (line 54) or an already existing one in the vertices set (line 60). If it is a new one, it is added to the vertices set (line 58) and to the queue (line 59) without adding edges. In the case where *copy* is in the vertices, no changes will be added. As with the FT case, the paths inherited from *curr* are set (line 63).

Ensuring Forward Compatibility

In the context of business protocol evolution, forward compatibility refers to the ability for clients of active conversations to continue interacting correctly with a given service after it is migrated to the new protocol (Ryu et al., 2008). Such a concept is crucial for conducting change impact analysis. Based on our previous framework articulated on the RONs structures and their associated reachability graph, we present below an algorithm for ensuring forward compatibility of active instances.

```
Algorithm 2: FComp (forward compatibility)
Input: List⟨Place⟩ps, Petri net, List⟨Rule⟩ chain
Output: decision on forward compatibility of instances
begin
Petri copy: = net.copy();
for (Rule r: chain)do
r.apply(copy);
endfor
List <Place> nextParam: = new ArrayList<>();
for (Place p: ps) do
List <Transition> ts: = new ArrayList<>();
net.getTransitions().forEach(t->{boolean exist = t.getPre().
stream().anyMatch(l->l.getP().getName().equalsIgnoreCase(p.
getName()));
if (exist)then ts.add(t);});
if (ts.isEmpty()) then
boolean comp: = copy.getPlaces().stream().anyMatch(pl->pl.
getName().equalsIgnoreCase(p.getName()));
if (!comp)then
return false;
```

```
endif
else continue;
endif
for (Transition t: ts) do
Transition tn: = copy.getTransitions().stream().filter(tr- >tr.
getName().equalsIgnoreCase(t.getName())).findFirst().orElse(null);
if (tn == null || !tn.getPre().stream().anyMatch(l->l.getP().
getName().equalsIgnoreCase(p.getName())))then
return false;
else
for (Link lnk: t.getPost()) do
if (!tn.getPost().stream().anyMatch(l->l.getP().getName().
equalsIgnoreCase(lnk.getP().getName())))then
return false;
endif
nextParam.add(lnk.getP());
endfor
endif
end for
end for
return nextParam.isEmpty(?true:isForwardComp(nextParam, net,
chain);
end
```

Description of the Algorithm Fcomp

This algorithm aims to check rules contained in the parameter *chain* when they are applied to the Petri net designated with *net*. First, the algorithm creates a copy of the original Petri net (parameter *net*) and applies all the rules in the list named *chain* (line 1), and the resulting modified net is called *copy* (lines 2-4). Now, the two nets (*net* and *copy*) are compared place by place and transition by transition. During this comparison, each place existing in the first Petri net *net* should also exist in the *copy* one. The same verifications are conducted for transitions of the two Petri nets. In the case in which the comparison induces a mismatch, the considered Petri nets (*net* and *copy*) are considered incompatible. However, not all places and transitions are examined. In fact, we are only interested in the marked places and all structures of the Petri net outgoing after the current position, i.e., forward compatibility. To do that, a third parameter named *ps* is introduced in the algorithm. It captures the marked places during the first call of the function, and it represents the list of places starting from the parameter net (line 6). At this stage, the Fcomp algorithm checks if each transition in *ts* exists in the Petri net *copy* (line 8). In the case in which this test is negative, the algorithm directly returns false (line 13). Otherwise, it gets all the POST places from it and adds them to a list (variable "List $\langle \text{Place} \rangle$ nextParam"). Now, if the algorithm terminates all the *ts* elements without returning any false, this means that the examined region in the two nets (*net* and *copy*) is a compatible one. Since it is the case, the algorithm continues checking the next region, which begins from the places contained in the list *nextParam,* and the algorithm operates a recursive call and restarts from this new list (line 31).

Ensuring Backward Compatibility

After having ensured migration to the new protocol, the backward compatibility class checks if the achieved backward path of an instance (also called *history path*) is compatible in the context of the new protocol (Ryu et al., 2008). Based on our previous modeling, we describe an algorithm for performing this compatibility type.

```
Algorithm 3 BComp (backward compatibility)
Input: History h, List <Rule> chain
Output: decision on backward compatible
begin
Petri cp: = h.getEnd().copy();
for (Rule r: chain) do
r.apply(cp);
endfor
for (Place m: cp.getMarked()) do
m.setTokens(0);
endfor
for (Place m: h.getStart().getMarked()) do
Place toM = cp.getPlaces().stream().filter(pl->pl.getName().
equalsIgnoreCase(m.getName())).findFirst().orElse(null);
if (toM == null)then
return false;
endif
toM.setTokens(m.getTokens());
endfor
int indS: = -1;
for (int i = 0; i < vertices.size(); i++) do
HighLevelPetri hlp: = vertices.get(i);
for (HighLevelPlace <Petri> hlplace: hlp.getNetHolders()) do
for (Petri p: hlplace.getElems()) do
if (p.equals(cp)) then
indS: = i;
break;
endif
endfor
if (indS >= 0)then
break;
endif
endfor
if (indS >= 0)then
break;
endif
endfor
HighLevelPetri st: = vertices.get(indS);
Petri startCopy: = h.getStart().copy();
Map <String, List<Rule>> trRule: = st.transRuleMap();
for (String ring: h.getTransSeq()) do
String[] trs: = ring.split(SEP);
if (trs[0].startsWith("TT")) then
Rule toEx: = trRule.get(trs[0]).get(0);
toEx.apply(startCopy);
else
startCopy.getTransitions().stream().filter(tr->tr.getName().
equalsIgnoreCase(trs[0])).findFirst().get().fire();
endif
List <String[]> arcs: = edges.get(st.getName());
```

```
String[] arc: = null;
for (String[] a: arcs) do
if (a[0].equalsIgnoreCase(ring)) then
arc: = a;
break;
endif
endfor
if (arc = = null)then
return false;
endif
st: = vertices.get(Integer.parseInt(arc[1].substring(3, arc[1].
length())));
for (HighLevelPlace <Petri> hlplace: st.getNetHolders())do
if (!hlplace.getElems().isEmpty()) then
List <Place> nMark: = hlplace.getElems().get(0).getMarked();
for (Place m: startCopy.getMarked()) do
Place nm = nMark.stream().filter(pl->pl.equals(m)).findFirst().
orElse(null);
if (nm == null || nm.getTokens() != m.getTokens())then
return false;
endif
endfor
endif
endfor
end for
return true;
end
```

Description of the Bcomp Algorithm

The Bcomp algorithm aims to ensure that a sequence of transitions expressing the historical achieved activities in the former protocol is still available in the new specification of the Petri net after its improvement with the list of rules called *chain*. First, the evolved Petri net called *cp* is obtained by making a copy of the current instance in the history (called *h.getEnd()*; lines 2–4) and after having applied the rules set (*chain*; line *1*). Now, the generated reachability graph is explored in order to search the variables *vertices* and *edges* in the initial Petri net leading to *cp*. If this exploration results in a positive conclusion, then the obtained sub-net is called *st* (line 33). In this case, we capture the real start instance from *h* (called *startCopy*; line 34). At this stage, two available nets are available: the real start instance *startCopy* and the new protocol start instance *st*. The algorithm takes the transition sequence of the history (variable *h.getTransSeq()*) and applies it to *startCopy* one by one (line 36). After each activation of the transition *t* to *startCopy*, the marking of this last one is modified, and it is compared to the child *st* coming from the edge *t* (from variable *edges*; line 44). If this comparison leads to a difference between the examined two paths, it is induced that the new protocol loses some places or transitions having already existed in the former version. In such a situation, a *false* conclusion is produced in the output, and the algorithm terminates its execution. Otherwise, the algorithm continues its progression by handling the whole transition sequence until it encounters an incompatibility situation, and it returns a *true* result (line 68).

The previous algorithms allow filtering migratable and non-migratable instances. Furthermore, they allow extracting the corresponding place reached by each migratable instance after its transfer to the new protocol. These places are called *replaceable places*. Since the name of the corresponding places may be altered by the service manager, a possible solution to this problem is suggested by Ryu

et al. (2008). This can lead to finding a place with an adequate corresponding name or concluding that the manager maps the new corresponding place. In the absence of the aforementioned scenarios, it is concluded that the considered place was deleted in the new process after having operated its evolution.

Handling Non-Migratable Instances

As was argued previously, active instances of an evolved protocol cannot be migrated to the new protocol version. Following the filtration of instances, satisfying forward and backward compatibilities by exploiting the previous algorithms, the remaining instances that cannot satisfy the compatibility types are managed in a special fashion. Using the ad hoc protocol is one of the most known approaches in the literature to manage non-migratable instances. Protocol managers must specify ad hoc protocols to ensure instances continuation and allow bridging the gap between the need for specifying the occurring changes per the new protocol and aims to guarantee the migration of a broad spectrum of active instances. By using ad hoc protocol, active non-migratable instances can continue their interaction on the evolved protocol as they are interacting with the old one. However, specifying ad hoc protocols is a challenge that needs to capture both protocols' similarities and mismatches. Nevertheless, based on our framework articulated on RONs for representing service protocols and their associated pushout rules for managing changes, the concern of specifying ad hoc protocols is approached in a more formal manner. In this sense, ad hoc production rules are conceived initially and then are used to produce the related ad hoc protocol. In fact, modeling and specifying ad hoc protocols is reduced to managing ad hoc production rules, and it is achieved transparently.

Figure 13 below depicts an ad hoc production rule. As it is shown in the figure, this production rule specification stipulates that the transition T13 (*report medical examination*) and the place P14 (*confirmed*) are inserted before the final place P8 (*processed*) by using a double pushout rule. By doing that, a gateway is established in the ad hoc protocol description. It ensures avoiding the production of future execution exceptions and run-time errors.

The previous pushout rule aims to handle service evolution requirements of the initial protocol of the Australian visa application and the migration needs for non-migratable instances. More precisely, it is used to elaborate the ad hoc protocol. As it appears in Figure 14, the resulting ad hoc protocol will serve to ensure the continuation of non-migratable instances of the initial Australian visa protocol in Figure 4.

RESULTS

This section exposes the results of the implementation of our approach. We start by presenting various scenarios describing the simulation of the Australian working visa application protocol and its

Figure 13.
Ad hoc production rule for the Australian visa protocol

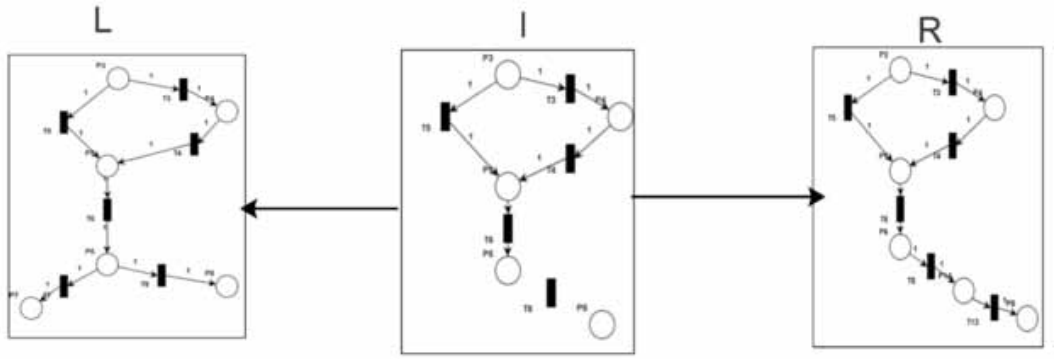
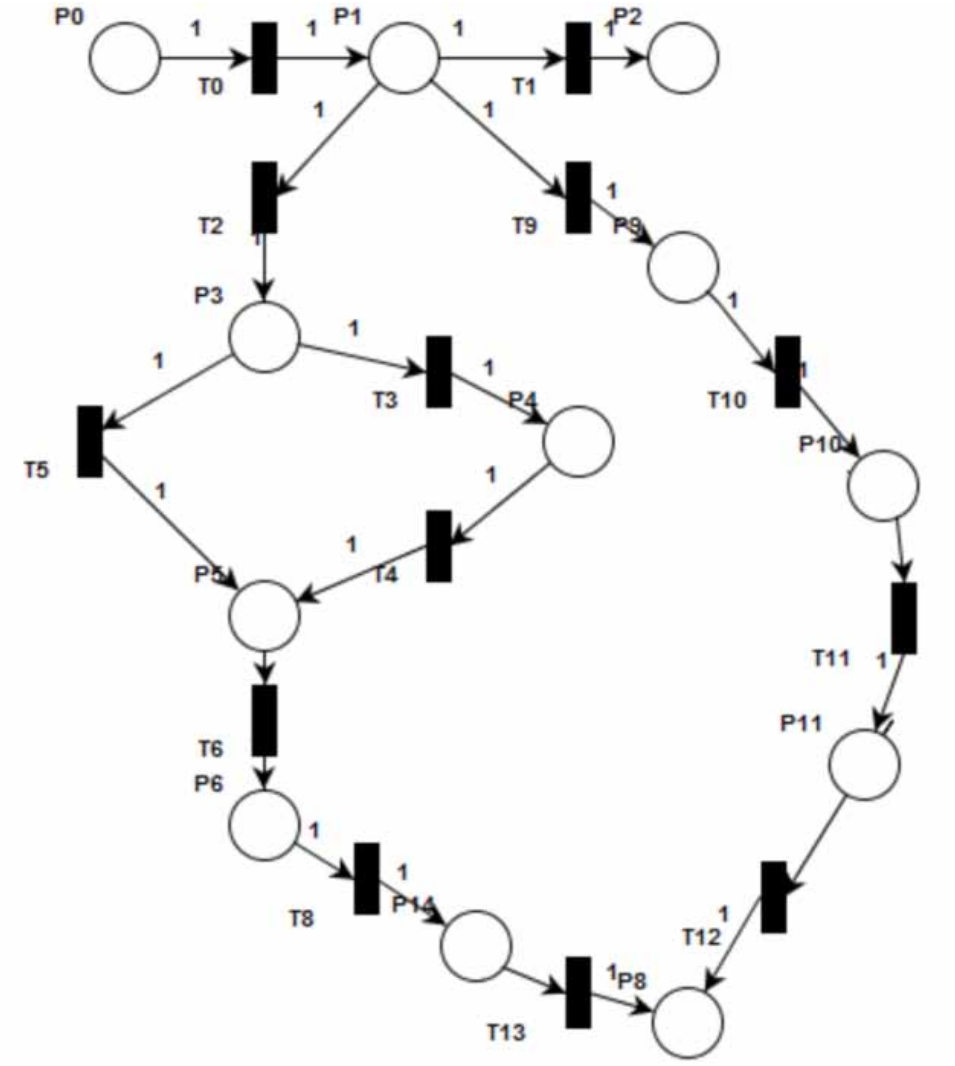


Figure 14.
The ad hoc protocol



evolution. Also, we depict some migration results. The Australian visa application protocol used in this paper and its related transformation rules were implemented and modeled in the RON-Editor tool as Petri net. As shown in Figure 15, the transformation rule $R1$ triggered the reconfiguration process of the token-net $TN1$ of Figure 4 to the token-net $TN2$ of Figure 9. Once the transformation rules were applied to the initial Petri net, the resulting token-net $TN1$ was obtained, as shown in Figure 16. Moreover, Figure 17 below shows the system-level net implemented in the RON-Editor.

Once the Australian working visa protocol was implemented in the RON-Editor and adequate transformations were performed, the active instances migration process could be triggered. Then experimental results were conducted. Furthermore, Figure 18 represents the results of the management of non-migratable (incompatible) instances.

```
vertex count: 38 \% we have 38 vertices
edge count: 35 \% we have 35 edges
```

Figure 15.
The transformation rule $\mathbf{R}1$

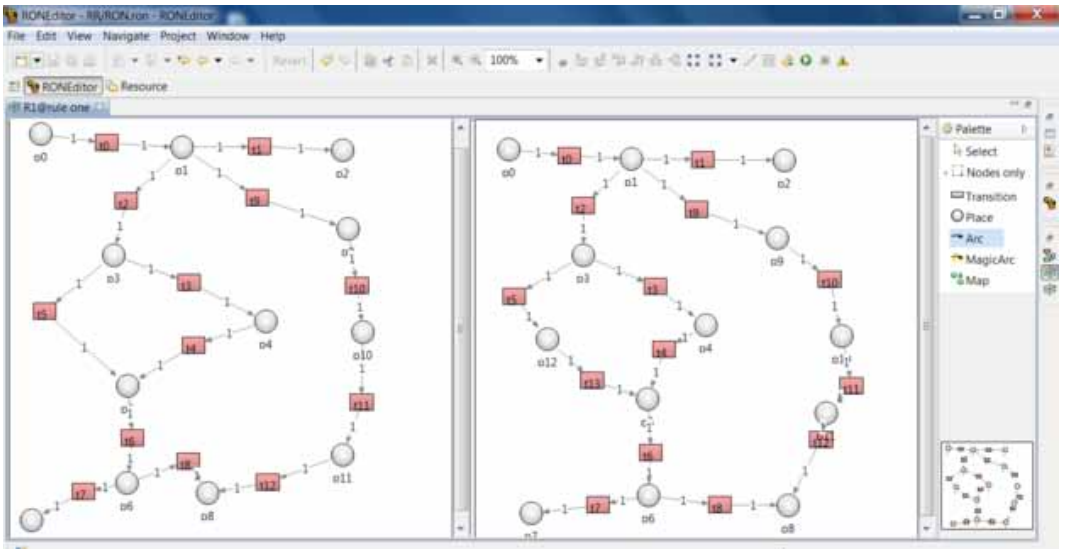
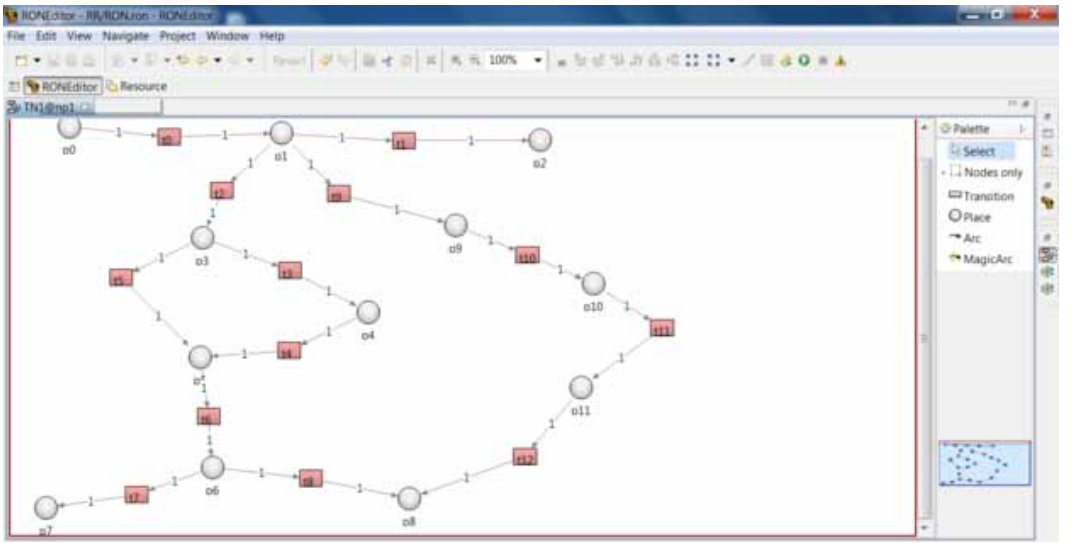


Figure 16.
The Token-Net $\mathbf{TN}1$




```
(HLP0, TT1, HLP1)
(HLP0, FT1, HLP2)
(HLP1, TT2, HLP3)
(HLP1, FT2, HLP4)
         ...
(HLP34, FT3, HLP36)
(HLP35, TT2, HLP37)
```

Figure 17.
The system level net (the RON model)

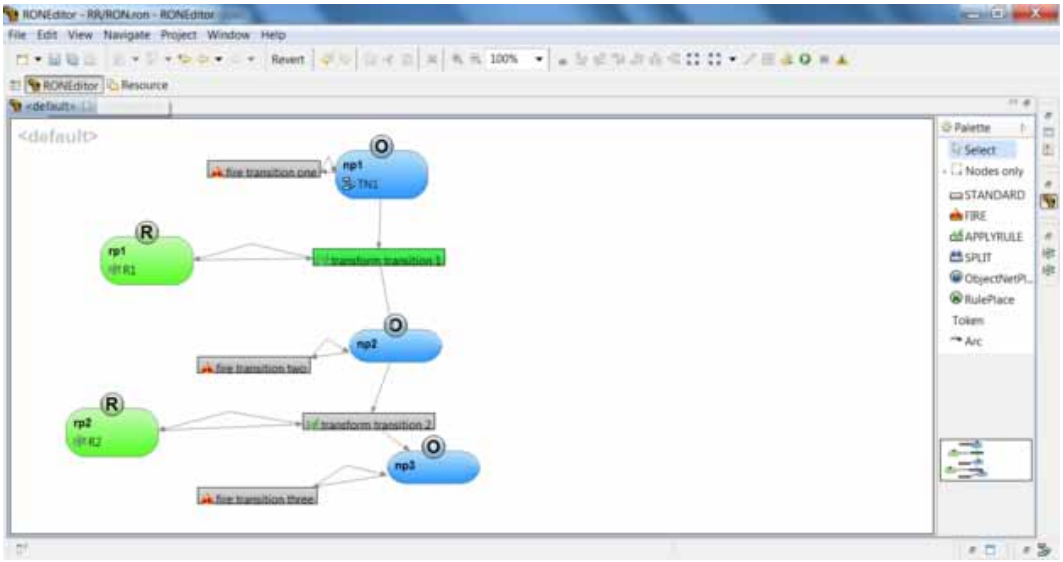


Figure 18.
Managing non-migratable (incompatible) instances

```
incomp size = 11
m = [P0:1,P1:0,P2:0,P3:0,P4:0,P5:0,P6:0,P7:0,P8:0,P9:0,P10:0,P11:0,]
trans path =
m = [P0:0,P1:1,P2:0,P3:0,P4:0,P5:0,P6:0,P7:0,P8:0,P9:0,P10:0,P11:0,]
trans path = T0,
m = [P0:0,P1:0,P2:0,P3:1,P4:0,P5:0,P6:0,P7:0,P8:0,P9:0,P10:0,P11:0,]
trans path = T0,T2,
m = [P0:0,P1:0,P2:0,P3:0,P4:1,P5:0,P6:0,P7:0,P8:0,P9:0,P10:0,P11:0,]
trans path = T0,T2,T3,
m = [P0:0,P1:0,P2:0,P3:0,P4:0,P5:1,P6:0,P7:0,P8:0,P9:0,P10:0,P11:0,]
trans path = T0,T2,T5,
m = [P0:0,P1:0,P2:0,P3:0,P4:0,P5:1,P6:0,P7:0,P8:0,P9:0,P10:0,P11:0,]
trans path = T0,T2,T3,T4,
m = [P0:0,P1:0,P2:0,P3:0,P4:0,P5:0,P6:1,P7:0,P8:0,P9:0,P10:0,P11:0,]
trans path = T0,T2,T5,T6,
m = [P0:0,P1:0,P2:0,P3:0,P4:0,P5:0,P6:1,P7:0,P8:0,P9:0,P10:0,P11:0,]
trans path = T0,T2,T3,T4,T6,
m = [P0:0,P1:0,P2:0,P3:0,P4:0,P5:0,P6:0,P7:1,P8:0,P9:0,P10:0,P11:0,]
trans path = T0,T2,T5,T6,T7,
m = [P0:0,P1:0,P2:0,P3:0,P4:0,P5:0,P6:0,P7:1,P8:0,P9:0,P10:0,P11:0,]
trans path = T0,T2,T3,T4,T6,T7,
m = [P0:0,P1:0,P2:0,P3:0,P4:0,P5:0,P6:0,P7:0,P8:1,P9:0,P10:0,P11:0,]
trans path = T0,T2,T5,T6,T8,
BUILD SUCCESSFUL (total time: 9 seconds)
```

```
HLP0: < <
NP1:[TN1:[P0:1,P1:0,P2:0,P3:0,P4:0,P5:0,P6:0,P7:0,P8:0,P9:0,P10:0
,P11:0,],],
NP2:[],
NP3:[], > >
----------------
HLP1: < <
NP1:[],
NP2:[TN1:[P0:1,P1:0,P2:0,P3:0,P4:0,P5:0,P6:0,P7:0,P8:0,P9:0,P10:0
,P11:0,P12:0,],],
NP3:[], > >
----------------
HLP2: < <
NP1:[TN1:[P0:0,P1:1,P2:0,P3:0,P4:0,P5:0,P6:0,P7:0,P8:0,P9:0,P10:0
,P11:0,],],
NP2:[],
NP3:[], > >
----------------
HLP3: < <
NP1:[],
NP2:[],
NP3:[TN1:[P0:1,P1:0,P2:0,P3:0,P4:0,P5:0,P6:0,P8:0,P9:0,P10:0,P11:
0,P12:0,],], > >
         …
HLP35: < <
NP1:[],
NP2:[TN1:[P0:0,P1:0,P2:0,P3:0,P4:0,P5:0,P6:0,P7:1,P8:0,P9:0,P10:0
,P11:0,P12:0,],],
NP3:[], > >
----------------
HLP36: < <
NP1:[],
NP2:[],
NP3:[TN1:[P0:0,P1:0,P2:0,P3:0,P4:0,P5:0,P6:0,P8:1,P9:0,P10:0,P11:
0,P12:0,],], > >
----------------
HLP37: < <
NP1:[],
NP2:[],
NP3:[TN1:[P0:0,P1:0,P2:0,P3:0,P4:0,P5:0,P6:0,P8:0,P9:0,P10:0,P11:
0,P12:0,] > >
```

DISCUSSION

Change management discussions in the research literature are restricted to managing simple operations, consisting only of adding and removing states and messages (Azough et al., 2009; Benatallah et al., 2005; Liske et al., 2009; Papazoglou, 2008; Ryu et al., 2007, 2008; Skogsrud et al., 2007; Y. Wang & Wang, 2013). Furthermore, only strict compatibility scenarios were addressed, and migration is conditioned by obligation rules that the running instances must satisfy to be migrated to the new protocol version. Thus, a major deficiency of the cited contributions consists of reducing protocol

compatibility to its obvious case (strict compatibility). Although Khebizi et al. (2017) proposed a declarative language to manage protocol evolution by specifying a set of migration patterns, the proposed framework appears very rigid and far from a concrete implementation.

In our approach, we ensure the junction between the formal aspect based on the RONs formalism and the practical aspect articulated on the reachability graphs and ad hoc production rules. On the one hand, old and new business protocols are modeled as Petri nets, and protocol evolution is perceived as a set of transformation rules. On the other hand, after protocol changes, RONs are used to handle the compatibility properties, and the reachability graph is exploited to ensure instance migration.

Furthermore, in our approach, we handle the sub-set of non-migratable instances that do not comply with the evolved protocol specification. Instead of using complicated ad-hoc protocols, as proposed by Benatallah et al. (2005), we deal with non-migratable instances by generating ad hoc production rules. We recall that in our approach RONs and the reachability graph are used as the main pillars to perform the progression of active instances according to the new protocol version. Notably, the analysis of the reachability graph of the RONs allows for effectively distinguishing migratable instances from non-migratable ones by specifying double push-out rules. The previously described algorithm RGGA allows for the construction of such a reachability graph, and the algorithms FComp and BComp are used to ensure, respectively, forward and backward compatibility properties.

From a practical point of view, as the RON-Editor allows basic functionalities for creating object nets and transformation rules, protocol evolution is expressed as a set of transitions (i.e., double pushout rules) to be fired in the RONs editor. Hence, once the reachability graph has been built, it is used to identify and transfer the migrating instances to the new version of the protocol. However, non-migratable instances are managed by deploying ad hoc production rules.

To evaluate the applicability of the proposed approach and to assess its relevance, we examine its effectiveness in a real-world scenario related to the Australian working visa application service. First, the associated service protocol is created in the RON editor as a Petri net model, and a synthetic instances dataset containing 5,000 instances was randomly generated and introduced as input of the system. Then, we operated various changes to the initial protocol description by adding or removing places and transitions. Such changes are expressed in the editor as transformation rules. After that, we deploy the previous reachability graph generator algorithm RGGA to generate the reachability graph of the visa protocol. Then, we analyze the impact of protocol changes based on backward and forward compatibilities properties and according to Fcomp and Bcomp algorithms.

The experimental results show that around 70% of active instances satisfy the backward or forward compatibility constraints and, consequently, are transparently migrated to the new protocol version. This important rate of migratable instances is justified by the low degree of change operations made on the protocol schema. However, for the remaining non-migratable instances (i.e., 30% of active instances), we apply the ad hoc protocol techniques by defining specific ad hoc production rules in the RON editor. For more flexibility in the proposed framework, the protocol manager enabled to define and customize his own context-specific migration rules that map old instances to new ones. This task is accomplished by specifying ad hoc production rules to be exploited in the RON editor for managing non-migratable instances.

We are convinced that giving the possibility to users to specify their own migration rules reinforces and improves the proposed conceptual framework. Indeed, more flexible migration strategies can be developed and configured by users when managing instances migration. It is worth noting that even though we consider only the business protocols of the Australian visa protocol during our experiment process, the principles proposed in our approach remain valid and can be applied to any other type of business process.

In this section, we discuss the results of our experiment. First, the screenshot in Figure 15 depicts two different parts. The first window, at the left of the figure, exposes the LHS expressing the Petri net $N1$, whereas the RHS represents the Petri net $TN2$. Furthermore, Figure 16 shows the net-places $np1$, the system net, the set of reconfiguration rules, as well as the object nets which were added by

the user. Notably, Figure 17, there are two rule-places $rp1, rp2$ containing, respectively, two token-rules $\left(R1, R2\right)$ as mentioned in the initial Figures 5 (a) and 5 (*b*). In addition, there are three net-places $np1, np2$ and $np3$ handling, respectively, the three token-nets $\left(TN1, TN2, TN3\right)$, as presented previously in (see Figures 4, 9, and 11). The screenshot also exhibits three fire-transition and a transform-transition 1 (in green color), which is enabled. It is worth noting that, as managing compatible instance migration can be operated with no particular difficulties, in our experimental process, the focus is made on incompatible ones needing specific attention.

Based on the forward and backward compatibilities principles, three classes of incompatible conversations are distinguished.

- Conversations with an incompatible history and a compatible forward path,
- Conversations with a compatible history and an incompatible forward path, and
- Conversations with both an incompatible history and incompatible forward path.

As it can be noted in Figure 18, an excerpt showing some migration results is depicted. The third group contains two separate execution paths. The first one expresses incompatible conversations with a transition path T0.T2.T5 and the second one is T0 T2 T5 T6.

CONCLUSION

Dynamic business protocol evolution is a crucial aspect in organizations evolving in dynamic and changing environments that are permanently constrained to update their business processes. Consequently, managing active instance migration after a business process evolution is a challenging issue for which both industrial and academic stakeholders have suggested many systematic approaches and techniques. In this work, a new approach based on RONs and their associated transformation rules are proposed to model service protocols and manage their evolution. The reachability graph of the evolved Net is constructed and exploited as a cornerstone to analyze the impacts of changes. The conceived framework allows handling instances migration to the new protocol version by exploiting the reachability graph of RONs. Furthermore, based on the Ron formalism and its properties, we can distinguish which conversations can be migrated to a new protocol when an old one has been changed. Finally, we thoroughly examined a real-world case study (the business protocol for the Australian working visa application service), and the approach is simulated and experimented with by using the RON-Editor tool.

As perspectives of our future work, we plan to deploy and implement the proposed approach to large scenarios of protocols while constructing suitable reconfigurable Petri nets in the context of business protocol evolution, such as e-commerce processes, e-learning, and e-health, and to analyze the related difficulties and impacts. Also, we plan to examine the performance of the conceived framework regarding the complexity of the process structure (the number of split and join constructs) and the number of active instances. As another possible orientation, we plan to focus on the data aspect by considering changes impacts on manipulated data and their consistency.

REFERENCES

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). *Web services*. Springer Berlin Heidelberg.

Azough, A., Coquery, E., & Hacid, M.-S. (2009). Supporting Web service protocol changes by propagation. *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, 438-441. doi:10.1109/WI-IAT.2009.76

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

Benatallah, B., Casati, F., Grigori, D., Nezhad, H. R. M., & Toumani, F. (2005). Developing adapters for web services integration. *Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005. Proceedings*, *17*, 415–429.

Benatallah, B., Casati, F., & Toumani, F. (2006). Representing, analysing and managing Web service protocols. *Data and Knowledge Engineering, 58*(3), 327–357.

Biermann, E., Ermel, C., Hermann, F., & Modica, T. (2007). A visual editor for reconfigurable object nets based on the ECLIPSE graphical editor framework. *Arbeitsberichte aus dem Arbeitsberichte aus dem Fachbereich Informatik*, 2.

Biermann, E., & Modica, T. (2008). Independence analysis of firing and rule-based net transformations in reconfigurable object nets. *Electronic Communications of the EASST, 10*.

Chang, V., Abdel-Basset, M., & Ramachandran, M. (2019). Towards a reuse strategic decision pattern framework: From theories to practices. *Information Systems Frontiers*, *21*(1), 27–44. Advance online publication. doi:10.1007/s10796-018-9853-8

Dam, H. K., & Ghose, A. (2015). Mining version histories for change impact analysis in business process model repositories. *Computers in Industry, 67*, 72–85.

Ehrig, H., Hoffmann, K., Padberg, J., Prange, U., & Ermel, C. (2007). Independence of net transformations and token firing in reconfigurable place/transition systems. In J. Kleijn & A. Yakovlev (Eds.), *Petri nets and other models of concurrency: ICATPN 2007* (Vol. 4546, pp. 104–123). Springer Berlin Heidelberg. doi:10.1007/978-3-540-73094-1_9

Ehrig, H., & Padberg, J. (2004). Graph grammars and petri net transformations. In J. Desel, W. Reisig, & G. Rozenberg (Eds.), Lectures on concurrency and Petri nets: Advances in Petri nets (pp. 496–536). Springer Berlin Heidelberg. doi:10.1007/978-3-540-27755-2_14

Ehrig, H., Padberg, J., & Ribeiro, L. (1994). Algebraic high-level nets. In H. Ehrig & F. Orejas (Eds.), *Recent trends in data type specification* (pp. 188–206). Springer Berlin Heidelberg. doi:10.1007/3-540-57867-6_11

Fdhila, W., Indiono, C., Rinderle-Ma, S., & Reichert, M. (2015). Dealing with change in process choreographies: Design and implementation of propagation algorithms. *Information Systems, 49*, 1–24.

Fedushko, S., Peráček, T., Syerov, Y., & Trach, O. (2021). Development of methods for the strategic management of Web projects. *Sustainability*, *13*(2), 742. doi:10.3390/su13020742

Fedushko, S., Ustyianovych, T., Syerov, Y., & Peracek, T. (2020). User-engagement score and SLIs/SLOs/SLAs measurements correlation of e-business projects through big data analysis. *Applied Sciences (Basel, Switzerland)*, *10*(24), 9112. doi:10.3390/app10249112

Graph Grammar Group. (n.d.). *AGG: The Homebase*. Retrieved August 10, 2020, from https://www.user.tu-berlin.de//o.runge/agg/index.html

Hoffmann, K., Ehrig, H., & Mossakowski, T. (2005). High-level nets with nets and rules as tokens. In G. Ciardo & P. Darondeau (Eds.), *Applications and theory of Petri nets 2005* (Vol. 3536, pp. 268–288). Springer Berlin Heidelberg. doi:10.1007/11494744_16

Hoffmann, K., Ehrig, H., & Padberg, J. (2009). Flexible modeling of emergency scenarios using reconfigurable systems. *Electronic Communications of the EASST, 12*.

Kahloul, L., Bourekkache, S., & Djouani, K. (2016). Designing reconfigurable manufacturing systems using reconfigurable object Petri nets. *International Journal of Computer Integrated Manufacturing*, *29*(8), 889–906. doi:10.1080/0951192X.2015.1130262

Kahloul, L., Chaoui, A., Djouani, K., Bourekkache, S., & Kazar, O. (2014). Using High Level Nets for the Design of Reconfigurable Manufacturing Systems. In ADECS@ Petri Nets (pp. 1-19). Academic Press.

Kaminski, P., Müller, H., & Litoiu, M. (2006). A design for adaptive Web service evolution. *Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems*, 86–92. doi:10.1145/1137677.1137694

Khebizi, A., Seridi-Bouchelaghem, H., Benatallah, B., & Toumani, F. (2017). A declarative language to support dynamic evolution of web service business protocols. *Service Oriented Computing and Applications*, *11*(2), 163–181. doi:10.1007/s11761-016-0204-7

Liske, N., Lohmann, N., Stahl, C., & Wolf, K. (2009). Another approach to service instance migration. In B. J. Krämer, K.-J. Lin, & P. Narasimhan (Eds.), *Service-oriented computing: ICSOC 2007* (Vol. 4749, pp. 607–621). Springer Berlin Heidelberg. doi:10.1007/978-3-642-10383-4_44

Llorens, M., & Oliver, J. (2004a). Introducing structural dynamic changes in Petri nets: Marked-controlled reconfigurable nets. In F. Wang (Ed.), *Automated technology for verification and analysis* (Vol. 3299, pp. 310–323). Springer Berlin Heidelberg. doi:10.1007/978-3-540-30476-0_26

Llorens, M., & Oliver, J. (2004b). Structural and dynamic changes in concurrent systems: Reconfigurable Petri nets. *IEEE Transactions on Computers*, *53*(9), 1147–1158. doi:10.1109/TC.2004.66

Mafazi, S., Mayer, W., & Stumptner, M. (2014). Conflict resolution for on-the-fly change propagation in business processes. *Proceedings of the Tenth Asia-Pacific Conference on Conceptual Modelling*, *154*, 39–48.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541–580. doi:10.1109/5.24143

Padberg, J., Ede, M., Oelker, G., & Hoffmann, K. (2012). Reconnet: a tool for modeling and simulating with reconfigurable place/transition nets. *Electronic Communications of the EASST, 54*.

Padberg, J., & Kahloul, L. (2018). Overview of reconfigurable Petri nets. In R. Heckel & G. Taentzer (Eds.), *Graph transformation, specifications, and nets* (Vol. 10800, pp. 201–222). Springer International Publishing. doi:10.1007/978-3-319-75396-6_11

Papazoglou, M. P. (2008). The challenges of service evolution. *Proceedings of the 20th International Advanced Information Systems Engineering*, *5074*, 1–15.

Prange, U., Ehrig, H., Hoffmann, K., & Padberg, J. (2008). Transformations in reconfigurable place/transition systems. In P. Degano, R. De Nicola, & J. Meseguer (Eds.), *Concurrency, graphs and models* (Vol. 5065, pp. 96–113). Springer Berlin Heidelberg. doi:10.1007/978-3-540-68679-8_7

Ryu, S. H., Casati, F., Skogsrud, H., Benatallah, B., & Saint-Paul, R. (2008). Supporting the dynamic evolution of Web service protocols in service-oriented architectures. *ACM Transactions on the Web*, *2*(2), 1–46. doi:10.1145/1346337.1346241

Ryu, S. H., Saint-Paul, R., Benatallah, B., & Casati, F. (2007, January). A framework for managing the evolution of business protocols in web services. In *Proceedings of the fourth Asia-Pacific conference on Comceptual modeling-Volume 67* (pp. 49-59). Academic Press.

Skogsrud, H., Benatallah, B., Casati, F., & Toumani, F. (2007). Managing impacts of security protocol changes in service-oriented applications. *29th International Conference on Software Engineering (ICSE'07)*, 468–477. doi:10.1109/ICSE.2007.49

Tarnauca, B., Puiu, D., Comnac, V., & Suciu, C. (2012). Modelling a flexible manufacturing system using reconfigurable finite capacity Petri nets. *2012 13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, 1079–1084. doi:<ALIGNMENT.qj></ALIGNMENT>10.1109/OPTIM.2012.6231954

Wang, J., Yu, D., Ma, X., Liu, C., Chang, V., & Shen, X. (2020). Online predicting conformance of business process with recurrent neural networks. *Proceedings of the 5th International Conference on Internet of Things, Big Data and Security*, 88–100. doi:10.5220/0009394400880100

Wang, Y., & Wang, Y. (2013). A survey of change management in service-based environments. *Service Oriented Computing and Applications*, *7*(4), 259–273. doi:10.1007/s11761-013-0128-4

Weber, B., Reichert, M., & Rinderle-Ma, S. (2008). Change patterns and change support features: Enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering, 66*, 438–466.

Yellin, D. M., & Strom, R. E. (1997). Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems*, *19*(2), 292–333. doi:10.1145/244795.244801

Zhao, X., & Liu, C. (2013). Version management for business process schema evolution. *Information Systems, 38*, 1046–1069.

Ali Khebizi received his PhD in computer science from Badji Mokhtar Annaba University in Algeria. Currently, he is a senior lecturer and a member of the LabStic Laboratory at 8 May 1945 University of Guelma, Algeria. His research interests include databases, Web services, protocol modeling, and verification, as well as business process evolution, integration, and management.

Radja Hamli is a PhD student at University Constantine 2 - Abdelhamid Mehri, Algeria, and she is a researcher at the MISC Laboratory, at University Constantine 2, Algeria. Her research interests include web services, protocol modeling and business processes evolution.

Allaoua Chaoui is a full professor of Computer Science at University Constantine 2 - Abdelhamid Mehri, Algeria. His research interests lie in the areas of services -oriented computing, business processes, and big data.

Raida Elmansouri is a doctor of Computer Science at University Constantine 2 - Abdelhamid Mehri, Algeria. Her research interests include data analytics, services computing and cognitive computing.