Architectural method to design and control dynamic composite web services

Manel Amel Djenouhat*
Cédric Laboratory,
CNAM,
Paris, France
and
LIRE Laboratory,
TLSI Department,
Abdelhamid Mehri University,
PB 67A, Nouvelle ville Ali Mendjeli, Constantine, Algeria
Email: manel.djenouhat@univ-constantine2.dz
*Corresponding author

Faiza Belala
LIRE Laboratory,
TLSI Department,
Abdelhamid Mehri University,
PB 67A, Nouvelle ville Ali Mendjeli, Constantine, Algeria
Email: faiza.belala@univ-constantine2.dz

Kamel Barkaoui
Cédric Laboratory,
CNAM,
Paris, France
Email: kamel.barkaoui@cnam.fr

Abstract: Nowadays, Web services constitute the core technology of IT infrastructure that has emerged in response to a fundamental shift in the way enterprises conduct their business. A componentised model emerges as the natural architecture for Web services-based applications. Using Mop-ECATNets formalism (Meta-Open Extended Concurrent Algebraic Term Nets) a sort of high-level Petri nets we show, in this paper, how we can ensure the formal specification of the dynamic Web services and control their interactions as well as their dynamic composition. Furthermore, in order to formally verify and execute Web services-based systems specifications, we implement Mop-ECATNet model in Maude system using an MDA (Model-Driven Architecture)-based approach.

Keywords: service-oriented architecture; dynamic web service composition; Meta-Open ECATNets; MDA; Maude.


Biographical notes: Manel Amel Djenouhat is a PhD student in the Computer Science Departments of the University of Constantine 2, Algeria, and of the Conservatoire National des Arts et Métiers of Paris, France. She is advised by Faiza Belala and Kamel Barkaoui. Her research interests include software engineering and formal methods, in particular, model-based software development and automated software engineering.

Faiza Belala is Professor in the Department of Software Technology and Information System at the University Abdelhamid Mehri, Constantine 2. She also currently serves as Chief of the GLSD team of the LIRE Laboratory in the same University. She received her PhD in Computer Science from Constantine University in 2002. Her research interests lie in the general field of theoretical computer science, with special focuses on formal methods (based on Petri nets, Rewriting logic, Bigraphs), applied to several domains as Web services computing and architecture description language. Her research has been published in various international journals and conferences. She frequently serves as a program committee member for various international conferences and workshops.
1 Introduction

Architectural design of large Web service-based applications has always played a significant role in determining the success of these systems. The current recognition of the importance of Service-Oriented Architecture (SOA) would appear to signal the emergence of a more disciplined basis for architectural design particularly if it combines formal models. This will significantly improve our ability to construct effective applications: on one hand, the SOA constitutes an architectural style defining an interaction model between the service providers, the service consumer and the service broker, and, on the other hand, formal models, as those related to Petri nets, use existing theoretical results and analysis tools to control and verify these interactions. Specifically, a principled use of their integration can have a positive impact on at least the following aspects of the development of services-oriented software: Specification, Understanding evolution, Reuse, Formal analysis, etc.

Our work aims to propose a high-level Petri nets-based approach, respecting the architecture style defining a SOA. It gives a set of patterns and guidelines for creating and controlling dynamic composite services that, because of the separation of concerns between description, implementation, and deployment, provide unprecedented flexibility in responsiveness to new reuses.

Effectively, several approaches exist in the literature to deal with specification and analysis of complex dynamic Web services. We may classify them in two main classes. The formal model-based approaches and the business process based ones. But, little work has been interested to achieve this goal respecting the SOA style. The first category is based on mathematical formalisms, the most common are: Petri nets and their different extensions, such as recursive ECATNets (Kheldoun et al., 2015), Coloured and Temporal Petri nets (Zhang et al., 2008; Suzuki and Lu, 1989), Process algebra, such as CCS and π-calculus (Milner, 1989; Lucchi and Mazzara, 2007), and Automata (Lee, 1960).

The second category includes the business processes modelling languages, such as BPMN (Business Process Modelling Notation) (Ouyang et al., 2008) and UML activities diagrams (Bock, 2003) which are considered less or not formal approaches.

Although formal models have proven an effective means to specify and deploy Web services, providing a precise mathematic semantics for the Web services composition, they give limits when the service properties deal not only with causality of events, but also with data types, recursive and timed constructs and failure recovery mechanisms (Friedrich et al., 2010). Besides, their implemented tools are too generic, they are used regardless of the applied domain; they do not support all the inherent aspects of dynamic and complex Web services management.

In this work, we propose a refined dynamic Web services composition model. In this respect, we address the dynamic services composition issue using a layered Petri net-based model, called Mop-ECATNet (Meta-Open Extended Concurrent Algebraic Term Nets) (Latreche and Belala, 2014). Unlike other approaches, this model may ensure, in a natural way, the formal specification of the recursive composition of dynamic Web services. We argue that their Meta-level controls the Web services interactions and also their dynamic composition. The contribution of this paper is twofold. On one hand, we propose an extension of the Mop-ECATNet model given in Latreche and Belala (2014), refining its Meta-level by defining new structures and patterns for its control layer to better manage the dynamic reconfiguration of Web services-based systems. On the other hand, we propose to implement Mop-ECATNet models in Maude system by defining a Meta-transformation, translating automatically a graph-based specification of Mop-ECATNet in Maude modules via ATL and ACCELEO tools. This gives arise to exploit the formal verification and analysis tools of Maude.

Section 2 presents some existing works that are related to our approach. Section 3 motivates our work using a much known example of Travel Agency. Section 4 recalls the basic concepts useful for understanding the paper notations. Section 5 describes how Mop-ECATNet model is used to control dynamic Web services composition, illustrating the defined patterns through our application example. The automatic execution principle of Mop-ECATNets is presented in Section 6. We conclude with a summary of contributions and an overview of future research directions.

2 Related work

While the best known formalisms alone provide a specification to compose and constrain flows of Web service invocation, there is little to support their models implementation. Process algebras and Petri nets-based formal approaches for Web services aim to fulfil the requirement of a coordinated and collaborative service invocation specification without any attention or particular support for atomic and compensable transactions.
In the context of process algebras, we cite the work of Câmara et al. (2006), which has proposed a formal method based on CCS for Web service choreography. Liu et al. (2007) have also used CCS algebra to model and specify Web services, in order to reason about the composition behavioural properties. In Lucchi and Mazzara (2007), the semantics of BPEL orchestration language is specified using π-calculus; however, parts of BPEL as data management are not addressed in this work.

By the same way and in order to consider the exchange of data during interactions and Web services dynamic composition, further work based on more advanced algebra have been suggested. Salaün et al. (2004) proposed a Bidirectional transformation from LOTOS to BPEL and vice versa. This translation includes exception handling, and enables verification of temporal properties. In Dumez et al. (2008), the authors also used the LOTOS to model the composition of Web services and use the CADP tool (Fernandez et al., 1996) for the formal verification. Although the process algebra is well suited for describing complex systems, their textual notation makes them less legible than Petri nets or automata.

Petri nets are very popular in the area of Business Process Management (BPM) due to the variety of control flow process that can model (van der Aalst, 2003). To demonstrate the ability of Petri nets to model Web service composition, a rigorous translation of BPEL in Petri nets was presented by Hinz et al. (2005). This translation covers all control structures and BPEL communication actions. The resulting models were used to verify the BPEL process through the WofBPEL tool (Ouyang et al., 2005).

Similarly, Hamadi and Benatallah (2003) have proposed a Petri net model which is more expressive for specifying the composition of Web services with no distinction between involved data types that may be complex in the Web service area (exchanged messages).

The work of Zhang et al. (2008) has addressed this problem by modelling Web services and their composition using coloured Petri nets (Jensen, 1989). The same model was reused later in Chemaa et al. (2012) in order to complete its associating semantics using Maude modules. This was achieved thanks to the graph transformation tool ATOM.

By the same thought, we propose in this paper an alternative definition of Mop-ECATNets model, allowing using their Meta-level, represented by an ECATNet, to control the Web services interactions and also their dynamic and transactional composition. In addition, to facilitate the implementation of this model in Maude, especially for user not familiar with rewrite logic concepts, we propose an automatic Mop-ECATNets/Maude translation using Atlas and ACCEOLEO tools.

We will note that some existing work attempt to give solution for managing transactional Web services composition (Bhiri et al., 2005; El Hadad et al., 2010; Orriëns et al., 2003) but these contributions still limited in terms of using adequate formalisms and tools.

3 Motivating example

Nowadays, Web services composition allows answering the increasingly complex users’ needs, by combining several Web services into a common business process. Composition in this case is classified into two approaches: orchestration and choreography. In orchestration, composition schemes are defined by specifying coordinator internal behaviour. Whereas in choreography approach, all peers participate in composition scheme definition. Besides, Web services composition may be accomplished manually (or semi-automatically) by giving semantic suggestions that allow to effectively selecting services, or automatically without human intervention. In this case, it permits to efficiently select and integrate heterogeneous services on the Web at runtime, reducing thus costs and development time.

Other problems may arise owing to the constant change of the environment. This will impose to consider the flexibility requirements. We mean by flexibility here, the ability to take into account the need for change, not only at the conceptual stage, considering the specification language expressiveness, but also and, above all, during the execution process.

Through the following example (see Figure 1), we will identify some problems that may occur during the execution phase of this composite Web service: Travel Agency. We will notice that there exist independent business services: ‘Hotel booking’, ‘Flight booking’, etc. If we aim to compose them into a ‘Travel Agency’ service for example, several scenarios may be considered, but not all lead to a valid or meaningful composition. Indeed, some customer requests may require simultaneous intervention of more than one service to be performed. The best is to compose them into a single entity while allowing them to collaborate with each other.

To simplify the complexity of Web services composition, we have to distinguish among the used services, during their execution process, the composite service, also called the dynamic service and the partner services that interact with it. In our case, the Travel Agency constitutes the dynamic service, while the partner services are: the Book Airline, the Book Room and the Rent Car ones. So, if a partner service fails to achieve this composition, it is therefore necessary to provide new ones in order to replace it. This situation may occur in our example for instance:

- When booking a hotel room, a connection problem can be observed (Service failure),
- When any partner service takes a long time to issue a response (Deadlock),
- When a partner service returns a response which does not correspond with client request (Negative Answer).

In this dynamic behaviour of reconfigurable Web services, we emphasise that the vitality of a service and its possibility to be replaced are transactional properties that should be expressed at the composite Web service level. These
Based on the atomicity degree of each service transactions, we may identify atomic or business ones. Recursively, a service is said atomic (i.e. with the semantics of ‘all or nothing’) when it becomes an indivisible component and its transactions (set of operations) cannot be subject to decomposition. In contrast, business service provides a flexibility allowing its compensation. Indeed, service compensation can resort a system to its original state upon the occurrence of an error or a situation where the business transactions could not successfully proceed any further. See Table 1 for a better illustration of the relevant notations.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Transaction kind</th>
<th>Partner service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Booking hotel service</td>
<td>Atomic re-executable [three times]</td>
<td>Access or rejection</td>
</tr>
<tr>
<td>Request (set place)</td>
<td>Business compensable</td>
<td>Response (list of available rooms) or (unavailable service, deadlock, service failure)</td>
</tr>
<tr>
<td>Selecting a room from the list or aborting</td>
<td>Atomic</td>
<td>Booking confirmation</td>
</tr>
<tr>
<td>Booking airline service</td>
<td>Atomic re-executable [three times]</td>
<td>Access or rejection</td>
</tr>
<tr>
<td>Request (set itinerary)</td>
<td>Business compensable</td>
<td>Response (list of available flies) or (unavailable service, deadlock, service failure)</td>
</tr>
<tr>
<td>Selecting an airline from the list or aborting</td>
<td>Atomic</td>
<td>Booking confirmation</td>
</tr>
<tr>
<td>Rent car service</td>
<td>Atomic re-executable [three times]</td>
<td>Access or rejection</td>
</tr>
<tr>
<td>Request (set itinerary)</td>
<td>Business compensable</td>
<td>Response (list of available cars) or (unavailable service, deadlock, service failure)</td>
</tr>
<tr>
<td>Selecting a car from the list or aborting</td>
<td>Atomic</td>
<td>Reservation confirmation</td>
</tr>
</tbody>
</table>

In these situations, it is imperative to maintain the flexibility control of the Web service composition with the capacity to act dynamically in a reduced time. It is, therefore, necessary to provide new partner services in order to replace the failed ones and, also, very important to locate failure points. This is achieved through the control of service transactions.

The main objective of this work is to put into practice effective solutions based on sound existing formalisms able to manage all the previous situations in a dynamic and automatic way.

4 Background

4.1 Maude and rewriting logic

Marti-Oliet and Meseguer (2002) have introduced rewriting logic as a consequence of their work on the general logics to describe concurrent systems. Several languages based on rewriting logic were created, the most known are: CAFEOBJ (Diaconescu and Futatsugi, 2002), ELAN (Borovanský et al., 1998) and MAUDE (Clavel et al., 2002).

In rewriting logic, a dynamic system is represented by a rewriting theory $\mathcal{T} = (S, E, L, R)$ describing the complex structure of its states and the various possible transitions between them. In rewriting theory definition, $(S, E)$ represents an equational membership theory, $L$ is a set of labels and $R$ is a set of labelled conditional rewriting rules. These rewriting rules can be of the following form:

$$r : [t] \rightarrow [t'] \text{ if } C$$

where $r$ is a labelled rule, all the terms $[t], [t']$ and $C$ are $\Sigma$-terms and the conditions $C$ can be rewriting rules, membership equations in $T_{\Sigma}(\lambda)$ or any combination of both.

Given a rewriting theory, we say $\mathcal{T}$ implies a formula $t \rightarrow t'$ and we wrote $\mathcal{T} \vdash [t] \rightarrow [t']$ if and only if it is obtained by a finite application of the rewriting logic deduction rules, Reflexivity, Congruence, Replacement and Transitivity (Marti-Oliet and Meseguer, 2002).
Architectural method to design

The theoretical concepts of the rewriting logic are implemented through the Maude language (Clavel et al., 2002). Its objective is to extend the use of the declarative programming and the formal methods to specify and verify critical and concurrent systems. It regroups three types of modules mainly: the functional modules that define the static aspects of a system, they form a Maude sub-language (extension of OBJ3) based on the equational logic; the system modules specify the dynamic aspect of the system using the rewriting rules; as well as the objects-oriented modules to specify the objects-oriented systems.

A Maude program represents a rewriting theory, i.e. a signature and a set of rewriting rules. The computation in this language corresponds to the deduction in rewriting logic. Furthermore, it is implemented through a running environment, allowing prototyping and formal analysis of concurrent and complex systems. MAUDE offers other tools like a theorem prover and a model-checker (LTL).

4.2 Mop-ECATNet definition

Mop-ECATNets (Meta-Open Extended Concurrent Algebraic Term Nets) (Latreche and Belala, 2014) are a sound combination of Meta-Petri Nets and Open-ECATNets. They inherit flexibility of control from Meta-nets and data structure, concurrency and composability from Open-ECATNets (Latreche et al., 2011). Indeed, this model consists of two levels, the higher level, represented by an ECATNet (Bettaz et al., 1992), a class of high-level algebraic Petri nets, and the lower one represented by a set of interacting Open-ECATNets.

Open-ECATNets extend ECATNets by adding a set of interfaces places in order to model, in a compact manner, open systems able to interact with their environment. This formalism has been showed well suited to specify and execute Web services interaction models, but it was not able to manage their reconfiguration at run time and in process failure case.
The main contribution of the Mop-ECATNets formalism is the presence of Meta-level nets, able to control transitions of the Lower level nets, thanks to their Meta-places. Mop-ECATNets are also equipped with timing constraints: on one hand, all lower level transitions are forced to fire when they are enabled; firing in this case is an atomic action. On the other hand, tokens of some places are constrained by timestamp. This allows specifying, for some places, the processing time of requests, and for other ones, the requests adaptation timers. Besides, the Mop-ECATNets originality lies in their semantics behaviour expressed naturally in terms of rewriting logic. Formally, the Mop-ECATNet definition given by Latreche and Belala (2014) is as follows:

Definition 1: (Latreche and Belala, 2014) A Mop-ECATNet \( ME = \langle E, O, Q, \lambda \rangle \) is a Meta-transitional net having as higher level net the marked ECATNet \( E \) and as lower level the Open-ECATNet \( O \). \( Q \) is the incidence function mapping Meta-places of \( E \) to transitions of \( O \) and \( \lambda \). \( ST \rightarrow N \) is a function that maps each token of some state place \( \mu \) (belonging to \( O \)) to its recon definition 1:

- A module representing the conveyed tokens through system states (places),
- Another one representing the active transitions across which tokens are firing, and
- A module representing the different system configurations (initial, intermediate and final).

The Mop-ECATNet Meta-level that manages the reconfiguration of the system in case of failure is specified by rewrite rules. Running the example in Maude gives the following scenario:

1. The transition \( t \) is activated upon the application of a rewrite rule, the initial marking of the dynamic place \( P1 \) is decreased by 1 and a timed token (3 time/units) is added in place \( P2 \) (time denotes the duration of the processing).
2. A conditional rewrite rule is also associated with the transition \( t \). It is applied when the processing time is shorter than token lifetime.
3. The reconfiguration process is insured by the Meta-level. So, the transition \( t \) is controlled by the Mp Meta-place, in case of failure, the Meta-level intervenes and activates a rewrite rule allowing the firing of the \( mt \) transition and the Meta-place \( Mp' \) to consume the token from \( Mp \). The place \( Mp' \) controls the transition \( t' \) and allows performing a new solution to address this failure.

Thus, Maude provides a platform for an easy development and effective implementation of Mop-ECATNets. In the following, we will show how we integrate automatically this model in Maude, using an MDA (Model-Driven Architecture)-based approach, in order to formally verify and execute Web service-based systems specifications.

## 5 Complex and dynamic web services as Mop-ECATNet

### 5.1 The principle of our approach

The field of Web services is dynamic and evolving; new services can be added, existing services are constantly changed, temporarily suspended, or finally deleted. Volatility of Web services is reflected in the choice of participant Services (or alternative Web services) and this, at the composition execution time (dynamically), which can increase the chances of the composition validation.

The volatility of Web services, the variability and the dynamic change of Web services transactional properties make difficult, if not impossible, the static prediction of all scenarios that may arise during the composition process, at the design phase. We present in this work an alternative definition of the Mop-ECATNet model that will manage, most prominently, the control and flexibility of the dynamic composition of Web services in terms of their transactions (or operations) properties.

We define a composite Web service as a set of some Open-ECATNets. Each net models a given service that may fail to achieve the composition process. In this case, it can be replaced by an equivalent Web service providing the same functionalities. A Web service or its Open-ECATNet is said: compensated, if it provides a set of operations to acquire (if possible) definitely a resource, it also offers an offsetting transaction. So, a compensation process aims to cancel the effects of a Web service operation which could not be completed successfully. In the same context of composition failure, an Open-ECATNet of any participant service is Re-executable, if it is characterised by a finite number of allowed attempts and a waiting time between service execution attempts. Besides, operations or transitions of each Open-ECATNet may be vital or no, replaceable or no.

In order to provide flexibility to a composite Web service, our approach aims to act at two stages:

**The design stage:**

1. Give a clear separation between the kind of the intervening services and their relationships, providing new compact structures to reduce the graph complexity of the composition.
2. Refine the Mop-ECATNet model by introducing new generic notations, which provide a more abstract view identifying all dynamic composition key elements in upstream.
The behavioural stage:

1. Classify participant services according to their atomicity and vitality degree.
2. Improve the Meta-level control strategies (for instance, remove all Meta-edges linked unnecessarily to dynamic transitions, etc.).
3. Propose new reusable patterns for the control level.
4. Introduce the parallelism concept in the Meta-level to allow a flexible reconfiguration.

5.2 The flexibility control based Mop-ECATNet

Although the Mop-ECATNet model is graphic, the growth of service participants in a composition makes the specification more complex. We propose, in this section, to refine the control layer in this model in order to manage conveniently the reconfiguration and the failure in Web services dynamic composition. This new executable and refined composition model is staffed with control structures and patterns.

Table 2 summarises the mapping rules specifying dynamic and complex Web services with Mop-ECATNet.

Figure 3 Traditional Mop-ECATNet model for the Travel Agency example (see online version for colours)
We illustrate through our running example of the Travel Agency composite Web service the proposed design structures that allow having a compact and readable graphical specification of the Mop-ECATNet model. A list of services is first identified:

1. **D**: the dynamic Web service that requests a fly ticket, a hotel room and a car.
2. **P1**: the partner service to book a fly ticket.
3. **P2**: a second partner service to book a hotel room.
4. **P3**: the third partner service to rent a car.
5. **S**: the composite Web service, specifying the interaction of **D** with **P1**, **P2** and **P3**.

Then, in a second step, the service controller provides the lists of alternative services of the participant ones in order to invoke them in case of dynamic system reconfiguration, as for instance:

- **P1’**: an alternative service of **P1**, it books a fly ticket in case of the **P1** service fails.

Finally, the dependence relationships between services are highlighted.

Using the traditional definition of the Mop-ECATNet model, we give an ad-hoc specification of this example in Figure 3. It reflects the complexity of the composition graph representing, in this case, only three composed services.

Indeed, it is hard to distinguish between the Mop-ECATNet levels and we are forced to highlight each Op-ECATNet representing a participant service.

Obviously, we note that the complexity of the graph and the dependence of its various elements greatly limit their reuse. Hence, a clear separation is naturally made between the different levels of the composite Web service structure, to provide a more compact and comprehensible model, especially in dynamic composition involving a large number of Web services. The same example used through its graphical model in Figure 4 illustrates this contribution.

We observe in this figure that the structures offer an abstract view and better reflect the system behaviour.

The distinction between the two levels (Meta and low levels) is obvious and the relations between services and their mutual dependencies are systematically highlighted.

Formally, an alternative definition of Mop-ECATNet model is given in the following.

**Definition 2**: A Mop-ECATNet MEWS modelling a Web service WS is a tuple \(<C, S, A>\) where:

- **C** is the marked controller service,
- **S** is the composite service, such that: \( S = (D, R, P, P', CP, Pi, Ti) \), **D** is the dynamic Web service having a set of requests \( R \); **P** and **P’** are respectively finite sets of Partner services and alternative Partner services; **CP** is a set of service control points; **Pi** and **Ti** are respectively finite sets of interface places and transitions (with \( Pi \cap Ti = \emptyset \)).
- **A** is the set of arcs joining the two Mop-ECATNet levels at the service control points **CP**.

Till now, we have shown how to use the Mop-ECATNets as a formal specification support for defining the structural aspect of the complex services composition. In the following, we will be closely interested by how to enrich the control level of the Mop-ECATNet model in order to ensure that the behaviour aspect of the configuration and reconfiguration is conform to the behaviour expected by dynamic services in their composition. In this formalisation part, the control model should present all possible branches and scenarios, explored at run time to reach a reliable system reconfiguration. Table 3 summarises some additional features, proposed to refine the control layer of a Mop-ECATNet.

**Table 3** Mop-ECATNet control layer extension

<table>
<thead>
<tr>
<th>Behaviour of dynamic controlled service</th>
<th>Mop-ECATNet-based formalisation</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic transaction</td>
<td>Unbound state (low-level transition)</td>
<td>start (\rightarrow) end</td>
</tr>
<tr>
<td>Business transaction</td>
<td>Controlled state (Meta-edge linked to the low-level transition)</td>
<td>Interruption</td>
</tr>
<tr>
<td>System reconfiguration</td>
<td>Rewrite rules</td>
<td><img src="image" alt="Rewrite rules" /></td>
</tr>
</tbody>
</table>

According to the Mop-ECATNet control layer extension, we propose a set of patterns to formalise the dynamic
A combination of the above rewriting rules.

Definition 3: Given $ME_{WS}$, a Mop-ECATNet modelling a composite service, strategies set defining its behaviour is given by $S_{ME_{WS}}$, recursively formed as follows:

- A strategy $S \in S_{ME_{WS}}$ is a simple rewriting rule representing atomic transaction, or
- A finite sequence of rewriting rules defining a compensable business transaction, or
- A set of rewriting rules expressing executable service operations, or
- A set of coordinated conditional rewrite rules representing the direct or/and the indirect dependency synchronisation patterns, or
- A combination of the above rewriting rules.

First, we have to identify some transactional properties (atomic and business transactions) of the given Web services. Then, some Petri nets structures have to be adjusted in order to respect these properties, for instance: (1) we remove Meta-edges linking dynamic transitions to Metaplaces, knowing that atomic transaction cannot be compensated, and (2) we introduce parallel transitions in the control layer to avoid the unnecessary reconfiguration of all partner services. Also, it is more judicious to conceive a set of reusable patterns to be directly applied in some known situations avoiding thus redundancy in complex services composition.

Indeed, services are running in a parallel way and may depend on each other. The dependencies between services require the dynamic synchronisation of the concerned services at specific points during the execution of their respective processes.

There are two types of dependencies, direct dependencies and indirect ones.

The IDP and DDP patterns are reusable and dedicated to deal with two specific synchronisation cases.

Some dependencies between services do not interrupt the execution processes of the concerned services.

This dependency refers to the indirect dependency; the IDP pattern is implemented to manage dynamic synchronisation at specific points outside the execution context of each process (non-blocking synchronisation).

The pattern uses the concept of appointments (rendezvous) to synchronise processes.

For instance, to ensure the airport transfer (from the airport to the hotel) in the Travel Agency example, both the hotel and airport addresses are required to invoke the rental car service. The Meta-level must thus handle the synchronisation of the two services at the point $t13$ (Figure 3) and ensure that their respective processes are correctly achieved.

On the other hand, to avoid deadlocks that can occur in case of direct dependency between services, we use the concept of semaphore (DDP pattern).

Direct dependency between services involves the interruption of one or more processes until synchronisation; the dependent process is stopped by the controller until the other processes reach the breakpoint. The controller reports this breakpoint with a semaphore to inform all the concerned entities.

Booking the hotel room service can have a direct dependency on the fly-ticket service; in fact, the fly ticket can be booked without constraint while the arrival date is necessary to book the hotel room. It is the Mop-ECATNet control layer’s task to abort the hotel service process until the arrival date is known.

### 6 Implementing Mop-ECATNet in Maude

This section aims to show how to provide an automatic generation of a Maude specification (the target model) from a Mop-ECATNet model (the source model), using a MDA transformation (Brown, 2004). This will be achieved by conceiving a tool support that interprets a given set of transformation rules.

Figure 5 presents a global view of our transformation approach; it describes the approach with its different abstraction levels: Metamodel level, model level, transformation level, and source code level. The transformation process produces Maude modules from the specifications contained in the Mop-ECATNet model. The transformation rules are defined upon the two designed Metamodels; the source and the target models which conform to their
Metamodels, and constitute respectively the input and the output of the transformation process. Our proposed transformation is exogenous and horizontal since both source and target models have the same abstraction level according to the classification presented by Mens and Van Gorp (2006). We may notice that we have implemented our transformation process thanks to ATL (Jouault et al., 2008) and ACCELEO (Musset et al., 2006) tools. The ATLAS Transformation Language (ATL) is a hybrid model transformation language developed as a part of the ATLAS Model Management Architecture. ATL is supported by a set of development tools built on top of the Eclipse environment: a compiler, a virtual machine, an editor, and a debugger.

**Figure 5** Mop-ECATNet2Maude transformation approach (see online version for colours)

An ATL transformation program is composed of rules that define how source model elements are matched and navigated to create and initialise the elements of the target models. By the other hand, ACCELEO is an open-source code generator from the Eclipse foundation that allows to use a model-driven approach to perform model-to-text transformation. Indeed, ATL transformation generates only models and ACCELEO continues the automatic transformation by associating the text code of the obtained (target) model.

### 6.1 Metamodels

#### 6.1.1 Source metamodel

Figure 6 shows the abstract metamodel defined for representing any Mop-ECATNet. We identify in our proposed metamodel the following classes:

- **Meta-level classes**: Meta-Place, Meta-Transition, Meta-Edge, Meta-Control-Edge, and Meta-Token.
- **Lower-level classes**: DPlace, DTransition, DEdge, Interface Place, and Token.

**Figure 6** The Mop-ECATNet metamodel (see online version for colours)

#### 6.1.2 Target metamodel

We were inspired from the Maude Metamodel published in Rivera et al. (2008), all Maude concepts are represented in its Metamodel (Figure 7). For our transformation approach, we use the following Maude classes: FModule, SModule, Sort, Statement, Equation, Condition, Operation, Rule and Term.
Architectural method to design

Figure 7  Maude metamodel (see online version for colours)

We note that the proposed source and target Metamodels are used as a syntactic reference for the introduced and the resulting models.

6.2 Transformation rules

In this section, we establish the set of correspondences between the two Metamodels respecting the syntax dictated by the ATL tool.

Table 5 enumerates some rules describing the correspondence between Mop-ECATNets and Maude Metamodel elements.

Table 5  Some transformation rules

<table>
<thead>
<tr>
<th>Rules</th>
<th>Mop-ECATNet metamodel</th>
<th>Maude-metamodel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place2Module</td>
<td>Place</td>
<td>Module Place</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>Supersort :Place</td>
</tr>
<tr>
<td></td>
<td>Sort</td>
<td>Arity:String</td>
</tr>
<tr>
<td></td>
<td>Capacity</td>
<td>Coarity:Place</td>
</tr>
<tr>
<td>Transition2Module</td>
<td>Transition</td>
<td>Module Transition</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>Supersort :Transition</td>
</tr>
<tr>
<td></td>
<td>TC</td>
<td>Arity:Transition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Coarity:Boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opn:_</td>
</tr>
<tr>
<td>Edge2Module</td>
<td>Edge</td>
<td>Module Edge</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supersort:Edge</td>
</tr>
<tr>
<td>MetaTransition2Subsort</td>
<td>MetaTransition</td>
<td>Subsort of Transition</td>
</tr>
<tr>
<td>Meta Place 2Subsort</td>
<td>MetaPlace</td>
<td>Subsort of Place</td>
</tr>
</tbody>
</table>

For example, the rule Place2Module bellow enables the transformation of a Mop-ECATNet place to a functional Maude module.

Rule Place2Module {
  From Source:Mop ECATNetMM!Place
  To Target: MaudeMM!ModulePlace (Name ← Source.Name)
}

It is composed of two mandatory (the from and the to parts) and is identified by a unique name (Place2module).

This rule specify the source model elements that must be matched (Place), the number and the type of the generated target model elements (Module Place), and the way these target model elements must be initialised from the matched source elements (Name). Thus, we define for each syntactic element of the Mop-ECATNet model its Maude-based semantics (see Table 5).

6.3 Mop-ECATNet2Maude tool application example

In order to illustrate our proposal, we reconsider the Travel Agency service (Figure 4).

To reach to the service Maude specification via the transformation approach, we create the Airline Web service source model from the Mop-ECATNet Metamodel (Mop.ecore). After conceiving the Metamodels plugins: Mop.ecore and Maude.ecore and giving all the corresponding transformation rules, we generate via the ACCELEO tool the output model as a Maude file.
In order to achieve a Maude textual specification from the graphical representation of the Travel Agency composition model, we proceed step by step, as follows:

- We give first, a sketch of the Travel Agency model based on the predefined structures in Table 1 and we get the abstract view of the model (Figure 4).
- Once the structure is valid, we use Tables 2 and 3 to define services deployment strategies (where we identify transactions, control points, patterns that may be used, etc.).
- We perform the first (model-to-model) transformation using the ATL tool.

For that, we need to provide:

- The Mop-ECATNet source Metamodel (Figure 6).
- The Maude target Metamodel (Figure 7).
- The Travel Agency source Model (Figure 3) in XMI which will be conform to the Mop-ECATNet Metamodel.
- The transformation model Mop-ECATNet2Maude.

The resulting XMI model is preserved for the next step of the transformation process.

- To obtain a Maude textual specification (in rewrite logic), we have to give a Maude template and a new transformation rules set according to the ACCELEO tool features.

Finally, the resulted Maude textual specification (see Figure 8) may be directly executed and checked via the existing tools around Maude system.

Figure 8 The Maude part code of the Travel Agency model

7 Conclusion

In this paper, we considered a kind of high-level Petri net for designing Web services composition models. Taking advantage of the powerful theoretical foundations of the Mop-ECATNet formalism, the proposed approach contributes to the development of modular, flexible and reconfigurable service-oriented system.

We enhanced and refined the model on both the structural and behavioural levels. At the structural level, we have provided new structures (more compact) to reduce the complexity of the graph composition by introducing a new grammar offering a more abstract view identifying, at the beginning, the key components of the composition.

At the behavioural level, we have extended the model control layer by defining new control strategies and implementing reusable patterns that lead to a recursive composition.

We have associated a Maude-based semantics to this proposed model by using a Meta-transformation technique, this allows to run and formal check the model, automatically on the Maude environment.

ATL and ACCELEO tools enabled this transformation (model-to-model and model-to-text transformations) and led us to obtain automatically a Maude textual specification from the initial composition graph.

This transformation has tended to make easier the specification for users unfamiliar with the Maude language since they automatically can get the source code from a graphical representation. It is also a time advantage for experts by allowing them to reach directly other phases such as checking system properties using Maude tools (e.g. model-checker).

In future, we aim to address other aspects (such as time constraints) and demonstrate the wealth of the Maude environment through its analysis and checking tools.

References


