ISSN 2278-3091
Volume 8, No.5, September - October 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse73852019.pdf
https://doi.org/10.30534/ijatcse/2019/73852019

Modeling and Formal Verification of Web Services Composition Using CADP

Nouha ADADI[1], Mohammed BERRADA[2], Mohamed HALIM[3], Driss CHENOUNI[4]

[1]IPI Laboratory, ENS-Fez, Sidi Mohamed Ben Abdellah University, Morocoo, nouhaadadi@gmail.com
[2]IPI Laboratory, ENS-Fez, Sidi Mohamed Ben Abdellah University, Morocoo, mohammed.berrada@gmail.com
[3]CMTSP Laboratory, ENS-Fez, Sidi Mohamed Ben Abdellah University, Morocoo, mohamed.halim@usmba.ac.ma
[4]IPI Laboratory, ENS-Fez, Sidi Mohamed Ben Abdellah University, Morocoo, d_chenouni@yahoo.fr

ABSTRACT

A growing number of companies are using web services to make their expertise and data available through the network. The current problem is the integration of these services with the aim of implementing business-to-business (B2B) collaboration. This task of collaboration is called the composition of web services. The main objective of this research work is proposing a new approach of modeling and verifying web services composition. This approach allows a clear and structured modeling and easy verification regardless of the number of services to be composed and the degree of complexity of composition. Concerning the contribution, we firstly propose a modeling of the composed system based on the multi agent systems and precisely the Multi-agent reactive decisional system (MARDS), and using the BPM (Business Process Management) standards in particular the BPMN notation. Secondly, we seek to verify this modeling to prove its correctness before implementing it. For verification, we propose a transformation of the BPMN model to a formal LOTOS specification, which has the advantage of being supported by formal verification tools such as the CADP toolkit, which allows to apply the behavioral properties and validate the system of Web services composition.

Key word: Web services composition; BPM; MARDS; formal LOTOS specification; CADP toolkit.

1. INTRODUCTION

The composition of web services is a very active area of research in the field of information technology, seeing the interest that it presents in terms of reuse of software components. However, the composition process is a complex global process. It consists of the processes of discovery, selection and coordination of services that must cooperate to meet a complex goal. Several works presented by the research community have focused on these different aspects of Web services composition. Among these works, we have the Models Driven Approach (MDA) define par l'OMG [1], which concentrates on the modeling phase. We propose an approach that combines the MDA approach with formal methods in order to reduce time and development costs, while making the composition of services more reliable.

The layout of this paper is as follows. In the second section, we present an overview of web services composition approaches as well as the cycle of development of our proposed approach. The third and fourth sections explain and detail respectively phases of modeling and of verification. The fifth section is devoted to the case study, we consider the online item purchase system as an illustrative example to apply the concepts of our approach. The conclusion and future work are presented in section VI.

2. PROPOSITION OF WEB SERVICES COMPOSITION APPROACH

2.1. Overview of web services composition approaches

Web Service composition is seen as a workflow design. The process of web service composition consists of creation of a workflow that realizes the functionality of a new service. In the literature, several approaches are proposed in order to compose web services; these approaches can be grouped into four classes: workflow-based approaches [2], approaches based on artificial intelligence planning techniques [3], approaches based on dependence graphs [4], and model-driven approaches [1]. The comparative study of these approaches leads us to develop the summary table 1.

Table 1: Qualitative comparison between web service composition approaches

Approaches	Techniques used	Dependence on the semantic model	Level of compos-ability	Verification Position
Workflow-based approaches	Software engineering techniques and workflow management	No	High	After implement-ation
Approaches based on artificial intelligence planning techniques	Artificial intelligence techniques	Yes	Medium	After implement-ation
Approaches based on dependence graphs	Theory of graphs and algorithms for optimal path search	Yes	Medium	After implement-ation
Model-driven approaches	Specification techniques such as modeling languages	No	Very high	Before implement-ation

The first three approaches are designed to meet the implementation phase of a composite web service; but they neglect the specification stage. This stage is very important because it allows to detach from the implementation to realize clearer abstract models, helping to the overall understanding of the system, and allowing to ensure that the system is meeting expectations. In addition, this specification is sufficiently expressive to allow for automated code generation.

Model-driven approach (MDA) concentrate on the realization of abstract models rather than on computer or algorithmic concepts. The specification phase is therefore particularly important in an MDA approach and represents a significant part of the development cycle. This allows developers to focus on the desired behaviour of the system, regardless of how to implement it. The partial generation of low level code from the specification, also reduces the time and therefore the development costs. For these reasons, we present a solution of web services composition faithful to the principles of MDA.

2.2. Development cycle

The main objective of MDA is the development of Platform Independent Model (PIM) from the Computation Independent Model (CIM), to allow the automatic generation of Platform Specific Model (PSM) and to obtain a significant gain in productivity. The transition from PIM to PSM involves model transformation mechanisms and a Platform Description Model (PDM). This approach is organized according to a development cycle "in Y". Our proposed approach of web service composition, consistent with MDA principles, follows the stages of this development cycle “in Y”. Figure 1 presents this cycle.

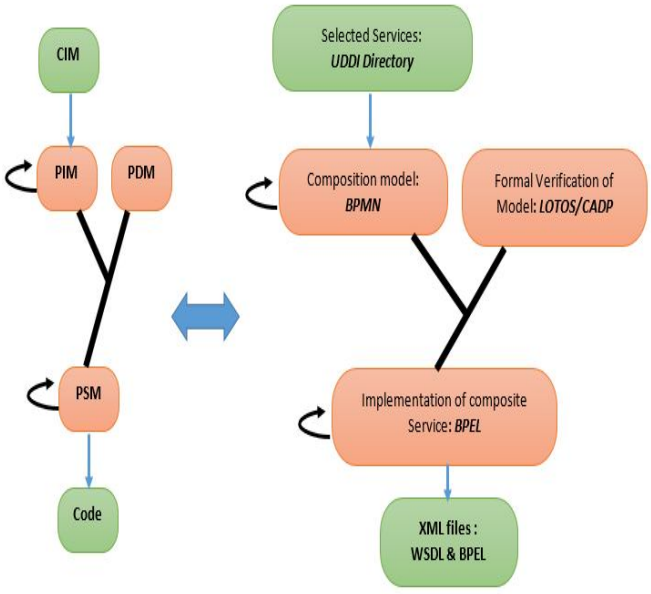


Figure 1: Proposed MDA approach to web services composition

The first step in the cycle is the discovery of services, using search mechanisms such as the UDDI directory. Once the existing services are selected, the second step is to achieve the composition model. In our approach, we adopt BPMN (Business Process Model and Notation) [5] as a modeling language. To prove that the composed service is performing the requested functionality, it is necessary to check its temporal properties. The third step is therefore based on the transformation of the BPMN models into a formal LOTOS description [6] in order to validate this later using the CADP tool [7]. The next step is the implementation of the system by generating BPEL [8] executable code from the BPMN specification. Finally, once the composed service is implemented, the last step is usually to publish it in the directory to facilitate its future use. The two stages of modelling and verification are presented in the following sections.

3. PHASE OF MODELING

The specification part consists in modeling the business processes of the composition of the web services. These processes can be described using modeling languages associated with the BPM [9] concept. This description becomes more difficult, complex and unstructured when the number of services is increasing, making the verification and implementation part difficult to afford. This problem requires providing a well-structured architecture that allows services to be composed in a simple but powerful way, and also allows you to add and remove services transparently without affecting other services. Multi-agent systems (MAS) are composed of autonomous agents that interact and coordinate to achieve their intentions. This makes them particularly suitable for composite and complex modeling of information systems. Some multi-agent models, such as the Multi-Agent Decision-Reactive System (MARDS) [10], have a well-structured hierarchical architecture and can be used to model business processes in a simple, powerful and transparent way to facilitate constraint checking and generation of executable code. This specification phase is therefore based on the MARDS model while using the BPM standards and especially the Business Process Management Notation (BPMN). It is called BPMN-MARDS profile. This profile is designed to improve the level of expressiveness of models in complex systems such as automated production systems, mobile systems [11] and the organization system [12]. This profile is customized for the domain of web services composition by adding to the models properties specific to the area of web services composition in order to facilitate their understanding as well as their transformation into code such as BPEL.

3.1. The Structure of the Services Composition Model

To present the specification phase we propose an abstract example of web service composition. In this example, we use 7 services (service1, service2, service3, service4, service5, service6, service7). Each of these services provides one or more methods and interacts with each other to respond to the customer’s request.

For modeling the example of Web Service composition we are going to follow these steps for composing MARDS agents:

- To organize agents in layers depending on the tasks and activities that they execute.

- To specify atomic agents which execute simple services, this agents present the basic components (DRA agents) of each layer.
- To identify the first and the intermediate composite agents (MARDS agent) of each layer, if they exist.
- To specify the principal composite agent that represents the agent which receives the main composite request from client.

Applying these steps to the example, we obtain the MARDS structure of this example shown in figure 2.

In this services composition model, the basic components are: "service2"; "service4"; "service5"; "service6"; "service7". The intermediate components are: “Amazon Ag”; “Pay Ag” and “service3”. The principal composite component is “service1”.

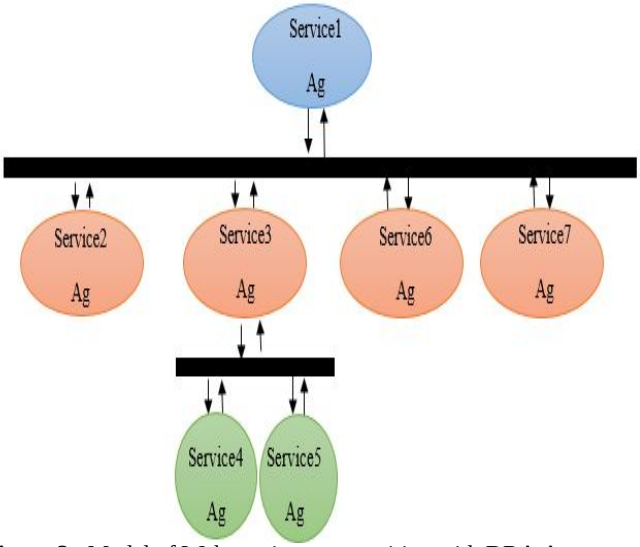


Figure 2: Model of Web services composition with DRA Agent

3.2. The Business Model of the Services Composition

Figure 3 shows the business model presented in the BPMN diagram of the web services composition. The action "A_service1" received by the component "service1" generates three decisions {D1_service1; D2_Service1; D3_Service1}. Each decision corresponds either to a sub-action received by a component such as the case of the components: "service2" {D1_service1, A_service2} and "service 3" {D2_service1; A_service3}, or to several parallel sub-actions received by different components as the case of the components: "service6" and "service7" {D3_service1, A_service6, A_service7}. Each sub-action received by any composite component will be realized and modeled as a sub-process.

The sub-action "A_Service3" received by the intermediate composite component "Service3" generates a sub-decision "D_service3". From its role this sub-decision generates two parallel sub-actions {A_Service4, A_Service5} for the components "service4" and "service5". The two sub-actions correspond to the sub-process of the sub-action "A_Service3".

Sub-actions {A_service2; A_Service4; A_Service5; A_Service6; A_Service7} received respectively by the basic components "Service2", "Service4", "Service5", "Service6", "Service7" generate the external states {Eext_service2; Eext_Service4; Eext_Service5; Eext_Service6; Eext_Service7}.

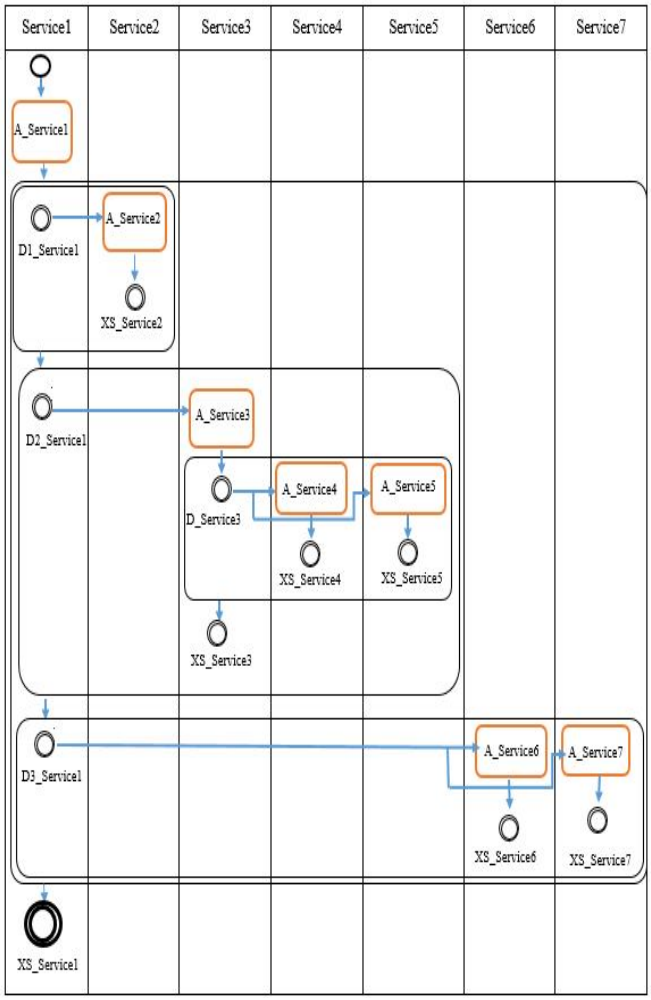


Figure 3: BPMN web service composition diagram

4. PHASE OF VERIFICATION

Formal verification is the systematic process of verifying, through exhaustive algorithmic techniques, that an implementation is in accordance with its specification. Using formal verification, all possible execution paths are analyzed mathematically without requiring the preparation of test cases. The developer describes simply the properties according to the system functionalities, which he wants to prove; and leaves the formal verification tools explore exhaustively all possible execution paths on the mathematical representation.

In this section, we describe the third phase of the development cycle of our approach, which is formal verification. This phase consists of two essential steps, the translation of the non-formal BPMN model into a formal LOTOS model, and the validation of this model using the CADP automatic verification tool. Figure 4 presents the different steps of the verification phase.

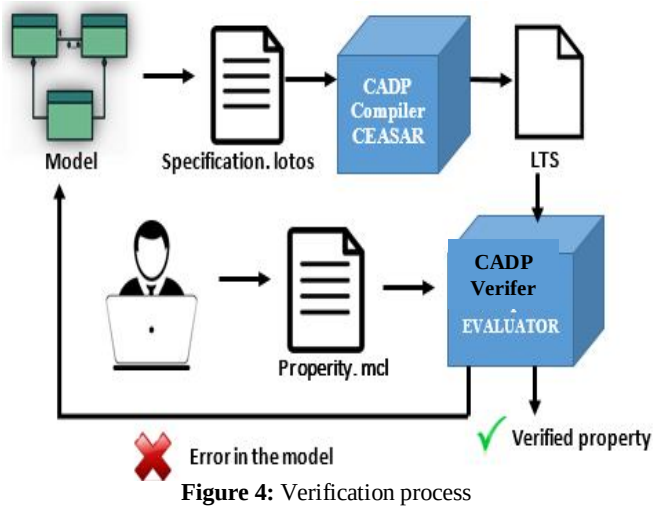


Figure 4: Verification process

4.1. Translation of the BPMN model into a formal LOTOS model

To provide the formal specification in LOTOS corresponding to the BPMN diagram presented in Figure 3, we follow the following steps:

- To define a process for each step of the activity, which means for each node of the diagram that can generate an activity. In the BPMN diagram shown in Figure 3, the processes are: "Initial", "process1", "process2", "process3", "process4", "process5", "process6", "process7", "Final". Each process is defined by a set of behaviors.
- To assign an identifier (integer) to each process. When sending a message through the INTERFi process, the issuer process provides the identifier of the destination process, its own identifier, a type of action and a possible parameter.
- To define the gates which are the communication channels between the processes. The gates in our approach are (SENDi, RECi) when i between 0 and n.
- To define operations between processes, in our example, all processes are executed at the same time using the ||| operator, which means that they are independent and do not communicate directly with each other, but they use INTERFi processes. The operator [| SENDi, RECi] | is used to synchronize the service processes with the INTERFi process via the ENVi and RECi gates, when i between 0 and n.
- To identify control flow patterns in the workflow to provide a definition (implementation) for each process.

```
Specification Specification_ Name [SEND, SEND0, SEND1, REC,
REC0, REC1]:noexit Behaviour
(Init [SEND, REC] (0) |[SEND,REC]|Process1 [SEND, REC ,
SEND0,REC0] (1)
(Process1 [SEND, REC , SEND0, REC0] (1)
||| Process2 [SEND0, REC0] (2)
||| Process3 [SEND0, REC0, SEND1, REC1] (3)
||| Process7 [SEND0, REC0] (7)
||| Final [SEND0, REC0] (8) ) |[ SEND0, REC0]|INTERF0
[SEND0,REC0]
(Process3 [SEND0, REC0, SEND1, REC1] (3)
||| Process4 [SEND1, REC1] (4)
||| Process5 [SEND1, REC1] (5)) |[ SEND1,REC1]|INTERF1
[SEND1, REC1] )
Where (* Implementation of processes *)
endproc
```

4.2. Formal verification with CADP

CADP (Construction and Analysis of Distributed Processes) [7] is a toolkit for the specification, rapid prototyping, verification, testing and performance evaluation of asynchronous systems. The tool incorporates a set of compilers and verifiers such as CEASAR and EVALUATOR.

We use the verifier EVALUATOR which support in the inputs a formal model and a set of behavioral properties. In the output it return the result (true or false) is that the property is checked or not in the model and a set of proposed correction. The behavioral property, defined by the developer, must be described as a formula of the temporal logic encoded in regular µ-calculus. As for the model, it must be provided in the form of a labeled transitions system (LTS). In our case, our model is expressed in LOTOS and CADP will have to compile it in order to obtain a mathematical representation in the form of an LTS. CADP will therefore use the CAESAR compiler to obtain the corresponding LTS, and then minimize it (if necessary) to improve its readability.

To perform these tasks we execute the following commands:

- Using CEASAR Compiler

```
$ /cadp/bin.win32/caesar.adt /cadp/fileName.lotos
$ /cadp/bin.win32/caesar /cadp/fileName.lotos
$ /cadp/bin.win32/bcg_min /cadp/fileName.bcg
$ /cadp/com/bcg_draw /cadp/fileName.bcg
```

- Using EVALUATOR verifier

```
$ /cadp/com/bcg_open /cadp/fileName.bcg
/cadp/bin.win32/evaluator /cadp/PropertyName.mcl
```

5. CASE STUDY: E-COMMERCE

As case study, we will consider in this paper an online item purchase problem. This is a simple illustrative example that present a typical scenario for web services composition problem. As far as creating the e-commerce composite service, we can use seven basic services ("Item", "Provider","Promotion","Cart", "Payment_Detail", "Bank" and "Transport") that will internally execute the e-commerce service, each one executes a set of tasks.

5.1. Modelling phase

A. Structure of Web services composition

The application of the concepts of MARDS model on our example allows to have the following structure (Figure 5) of the composition system by creating communication interfaces, new intermediate and main services.

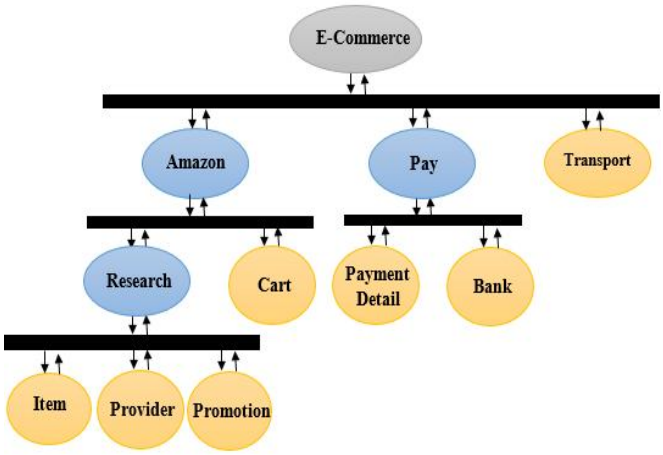
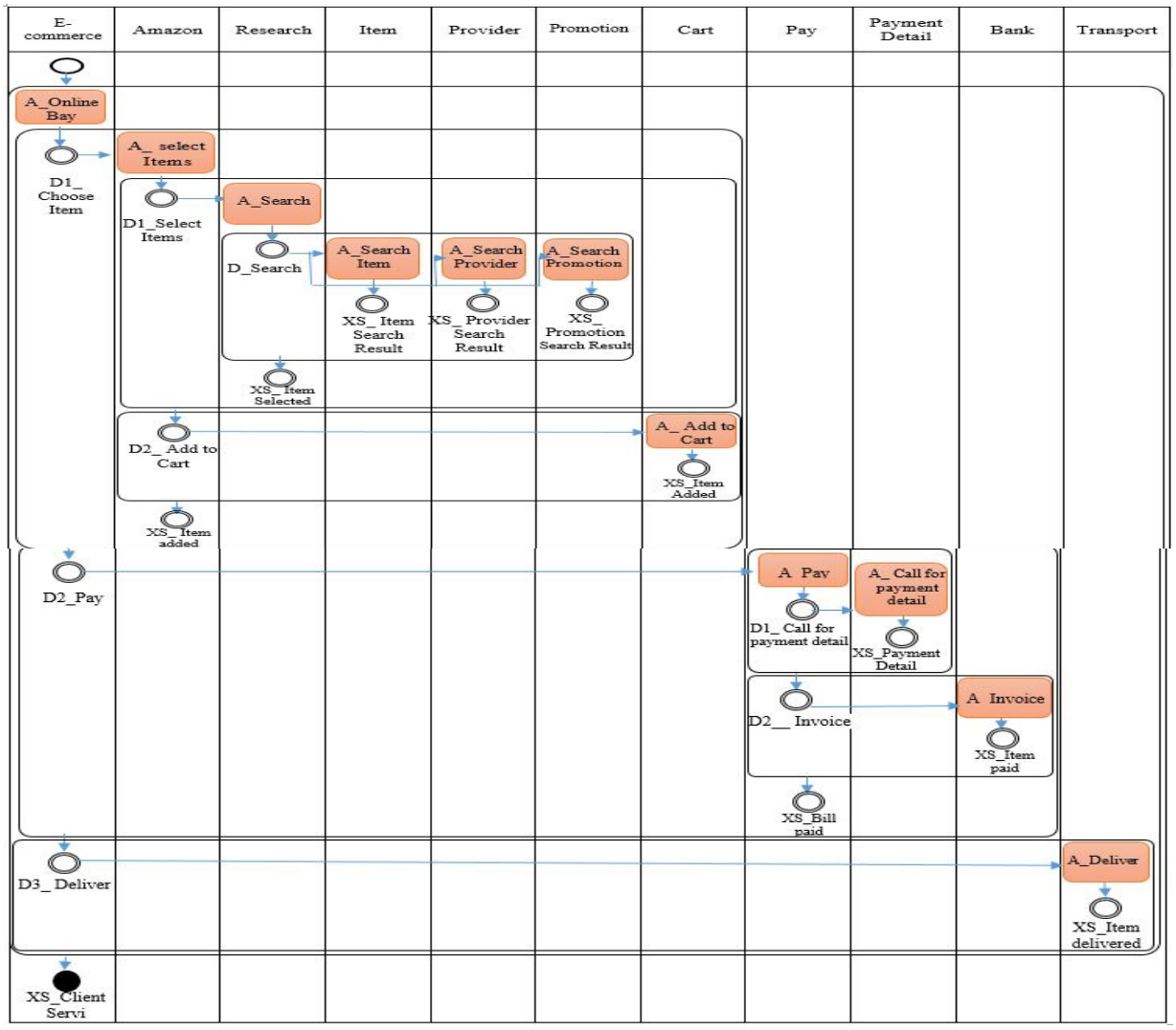


Figure 5: Web service composition model based on MARDS

In this model of service composition, the basic components are: ("Item", "Provider", "Promotion", "Cart", "Payment_Detail", "Bank" and "Transport". The intermediate components are: "Amazon"; "Pay"; "Research". The main composite component is "E-commerce".

B. Business Model of the Composite Service "E-commerce"

The Figure 6 shows the Business model of the composition services model based on MARDS. The action "A_Online Bay" received by "E-commerce" component generates three decisions {D1_Choose Items; D2_Pay; D3_ Deliver}. Each decision corresponds to a several sub-actions received by "Amazon" component {D1_Choose Items; A_Select Items}, by "Pay" component {D2_Pay, A_ Pay} and by "Transport" component {D3_ Deliver, A_ Deliver}. Every sub-action received by any composite component will be realized and modeled as a sub-process.



The sub-action "A_Select Items" received by the "Amazon" component generates two decisions {D1_Select Items; D2_ Add to Cart }. The first sub-decision "D1_ Select Items" generates the {A_ Search} action for "Research" composite component.

The second sub-decision “D2_ Add to Cart” generates the “A_Add to Cart” action for “Cart” basic component. The sequencing of the two sub-decisions corresponds to the sub-process of the “A_Select Items” sub-action.

The sub-action “A_ Search” received by the “Research” composite component generate the sub-decision “D_Search”. On his part, this sub-decision generates three parallel sub-actions {A_Search Item; A_Search Provider; A_Search Promotion} for the components "Item"; "Provider" and "Promotion". The three sub-actions correspond to the subprocess of the sub-action "A_Search".The sub-action “A_ Pay” received by the “Pay” component generates two decisions {D1_Call for payment detail; D2_ Invoice}. The first sub-decision “D1_ Call for payment detail” generates the {A_ Call for payment detail} action for “Payment_Detail” basic component. The second sub-decision “D2_ Invoice” generates the “A_ Invoice” action for “Bank” basic component. The sequencing of the two sub-decisions corresponds to the sub-process of the “A_ Pay” sub-action.

5.2. Phase of verification

A. Translation of the BPMN model into LOTOS specification

Applying the rules and methods described in Section 4 to the BPMN models presented in Figure 6, we obtain the following LOTOS specification.

```
Specification Online_Purchase [SEND, SEND1, SEND2, SEND3,
REC, REC1, REC2, REC3]:noexit
library
BOOLEAN, NATURAL, INTEGERNUMBER
endlib
type (*Definition of types*)
endtype
behaviour
(Init [SEND, REC]( 0 of Int)|[SEND, REC]| Ecommerce[SEND,
REC, SEND0, REC0] (1of Int))
|||(Ecommerce [SEND, REC, SEND0, REC0](1 of Int)
|||Amazon[SEND0,REC0,SEND1, REC1](2 of Int)
|||Pay[SEND0,REC0,SEND2, REC2](3 of Int)
|||Transport[SEND0,REC0](4 of Int))
|||final[SEND0,REC0](5 of Int)))|[SEND0,REC0]|
INTERF0 [SEND0,REC0]
|||(Amazon [SEND0,REC0,SEND1, REC1](2 of Int)
|||Research [SEND1,REC1,SEND3, REC3](6 of Int)
|||Cart [SEND1,REC1](7 of Int))
|[SEND1,REC1]| INTERF1 [SEND1,REC1] (<>)
|||(Pay [SEND0,REC0,SEND2, REC2](3 of Int)
|||Payment_Detail [SEND2,REC2](8 of Int)
|||Bank [SEND2,REC2](9 of Int))
|[SEND2,REC2]|INTERF2 [SEND2,REC2] (<>)
|||(Research [SEND1,REC1,SEND3, REC3](6 of Int)
|||Item [SEND3, REC3] (10 of Int)
|||Provider [SEND3, REC3](11 of Int)
|||Promotion [SEND3, REC3](12 of Int))
|[SEND3, REC3]| INTERF3 [SEND3, REC3] (<>)
where
(*Definition of process*)
process Init [SEND, REC] (Id:Int) : exit :=
Sequence [SEND, REC] (Id, 1 of Int)
>> exit
where
process Sequence [SEND, REC] (Emt_Id:Int, dst_Id:Int): exit :=
ENV !dst_Id !Emt_Id !RUN !void; exit
endproc
endproc
process Ecommerce [SEND,REC,SEND0,REC0] (Id:Int) : exit:=
REC ! Id ! 0 of Int ! RUN ! Void;
Sequence [ENV0,REC0] (Id,2 of Int)
>> Sequence [ENV0,REC0] (Id,3 of Int)
>>exit
where (*Definition of Sequence process*)
endproc
process (*Definition of Amazon process*)
endproc
process (*Definition of Pay process*)
endproc
process(*Definition of Transport process*)
endproc
process Research [SEND1,REC1, SEND3,REC3] (Id:Int) : exit:=
REC1 ! Id ! 2 of Int ! RUN ! Void;
ParallelSplit[SEND3,REC3](Id,insert(10 of Int,insert(11 of Int,
insert(12 of Int, emptyset))))
>>exit
where
process ParallelSplit [SEND3, REC3] (Emt_Id:Int,
dsts_id:IntSet) : exit :=
[empty(dsts_id)] -> exit
[]
[not(empty(dsts_id))] ->
(let dst:Int=pick(dsts_id) in
SEND3 !dst ! Emt_Id !RUN !void;
ParallelSplit [SEND3, REC3](Emt_Id, remove(dst, dsts_id))
)
endproc
endproc
process INTERF1 [SEND1, REC1] (B:Buffer) : noexit :=
SEND1 ?R:Int ?S:Int ?D:Cmd ?P:Int;
INTERF1 [SEND1, REC1] (B + Message (R, S, D, P))
[]
[not (empty (B))] ->
(let M:Msg = head (B) in
REC1 !getrcv (M) !getsnd (M) !getcmd (M) !getprm (M);
INTERF1 [SEND1, REC1] (tail (B))
)
endproc
process
(Definition of INTERF2 process)
endproc
```

```
process
(*Definition of INTERF3 process*)
endproc
Process Item [SEND2,REC2] (Id:Int) : noexit:=
REC2 ! Id ! 2 of Int! RUN ! Void;
stop
endproc
Process
(*Definition of basic process*)
endproc
process
final[SEND0,REC0](Id:Int) : exit :=
Synchronisation [SEND0, REC0] (insert(2 of Int, insert(3 of Int,
insert(4 of Int, emptyset))), Id)
>>exit
Where
process Synchronisation [SEND0, REC0] (Emts_Id:IntSet,
Id_dst:Int) : exit:=
[empty(Emts_Id)] -> exit
 []
[not(empty(Emts_Id))] ->
REC0 !Id_dst ?Emt_Id:Int !RUN !void [Emt_Id isin Emts_Id];
Synchronisation [SEND0, REC0] (remove(Emt_Id, Emts_Id), Id_
dst) endproc
endproc
endspec
```

B. Verification using CADP

After elaborating the LOTOS specification in the file "Ecommerce.lotos" we follow the following steps:

- To compile the Lotos specification "Ecommerce.lotos" by the Caesar.adt compiler to create the header file "Ecommerce.h" which is used to define the standard part of the specification. This part contains the abstract data, types, and algebraic operations of the specification.
- To compile the two files "Ecommerce.lotos" and "Ecommerce.h" with the Caesar compiler in order to create the file "Ecommerce.bcg", containing the tagged transitions system. To improve its readability, we can reduce this system without distorting the results using CADP's bcg_min application by minimizing many of their states and transitions.
- To visualize the labeled transitions system (LTS) using CADP's bcg_draw application.

The result of the execution of these steps is visualized in figure 7.

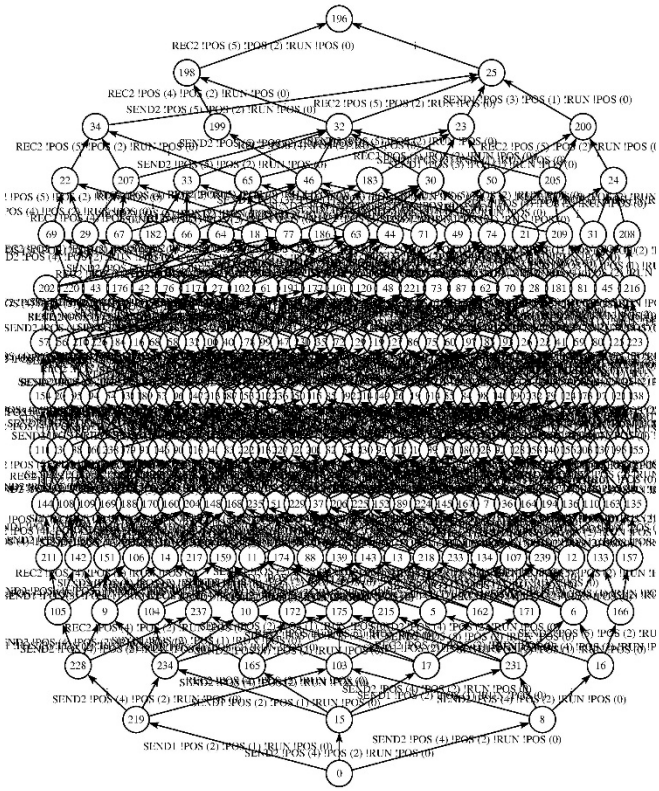


Figure 7: Labeled Transitions System (LTS) generated by CADP

After the creation of the labeled transitions system (LTS), the tool is ready to provide the verification results. The task of the developer here is to define, in a Property.mc file, behavioral properties using μ- calculus. This type of behavioral verification consists of the description of the expected behavior of the program, observed at a certain level of abstraction. In this type of verification, we can check the following two properties:

- **Safety**: This property wants to show that "a bad thing" will never happen. This property ensures the absence of states where the program has produced a wrong result (deadlock, errors, unsolicited response, and duplication of outputs). As part of our case study, for example, we want to prove that a client cannot view and download the payment details form (total, VAT, discount…) without selecting items and adding them to the cart.

```
macro Lead (A, B) =[true_.(A)]mu X.(<true> true and [not
(B)] X)
end_macro
macro SelectItems() = 'SEND1!POS(6)!POS(2)!RUN.*'
end_macro
macro Add to Cart() = 'SEND1!POS(7)!POS(2)!RUN.*'
end_macro
macro                    () = 'SEND2!POS(8)!POS(3)!RUN.*'
end_macro
Mener (not[                ], [SelectItems]false And [Add to
Cart]false)
```

- **Vivacity**: This property wants to show that "a good thing" must happen. This property includes the termination of treatments, assured response to requests, guaranteed occurrence of events and equity in access to a resource. As part of our case study, for example, we want to prove that a client can always perform the action pay by PayPal when the item or order is added to the cart and the payment details form (total, VAT, discount…) is posted.

```
macro Lead (A, B) =[true_.(A)]mu X.(<true> true and [not (B)] X)
end_macro
macro Add to Cart() = 'SEVD1!POS(7)!POS(2)!RUN.*'
end_macro
macro Payment_Detail () = 'SEND2!POS(8)!POS(3)!RUN.*'
end_macro
macro Pay by PayPal() = 'SEND2!POS(9)!POS(3)!RUN.*'
end_macro
Mener (Add to Cart and Payment_Detail, Pay by PayPal)
```

Finally we have to check these properties using the tool EVALUATOR. This later will mathematically explore all possible execution branches on the generated LTS to prove that the property is verified (or not).

The formal verification step can be repeated iteratively until a correct and refined composition model is obtained. The model can then directly and automatically transform into executable code without having to check the system after implementation.

6. CONCLUSION

In this work, an approach for specification and formal verification of composite web services is proposed.

The specification part insists on modeling the business processes of web services composition. These processes have been described using BPM standards and the MARDS model.

Our approach considers not only the specification but also the verification. Since it is preferable to detect errors as early as possible in the development cycle, from the specification stage we chose the formal verification of the models developed, because we consider that it is more reliable and easy. The developer does not need to achieve test sets and perform the simulation of system execution.

When the composition model is validated, the last step is the implementation, which consists in generating the BPEL executable code from the specification. This step will be the objective of a future work in which we will develop a framework that allows the automatic generation of BPEL code from the BPMN model.

REFERENCES

1. Richard Soley**, Model Driven Architecture (MDA)**, Draft 3.2. Object Management Group, Inc., November 2000.
2. Ardagna D, Comuzzi M, Mussi E, Pernici B, Plebani P 2007 Paws: **A framework for executing adaptive web-service processes,** IEEE Software 24 39–46. https://doi.org/10.1109/MS.2007.174
3. Lécué F, Léger A, Delteil A 2008 DL, **Reasoning and AI Planning for Web Service Composition,** Web Intelligence 445–53. https://doi.org/10.1109/WIIAT.2008.344
4. L. Ying, 2010, **A Method of Automatic Web Services Composition Based on Directed Graph**, cmc International Conference on Communications and Mobile Computing 1 527-31. https://doi.org/10.1109/CMC.2010.91
5. Chand, Donald & Chircu, A.M.. (2012). **Business process modeling**. 10.4018/978-1-4666-0249-6.ch003.
6. Lai R., Jirachiefpattana A. **Lotos In: Communication Protocol Specification and Verification.** The Springer International Series in Engineering and Computer Science, vol 464. Springer, Boston, MA. (1998). https://doi.org/10.1007/978-1-4615-5549-0_4
7. Garavel H, Lang F, Mateescu R, Serwe W, **CADP 2011: a toolbox for the construction and analysis of distributed processes**, International Journal on Software Tools for Technology Transfer, April 2013, volume 15, pp 89–107. https://doi.org/10.1007/s10009-012-0244-z
8. OASIS Standard. **Web services business process execution language**, version 2.0, April 2007.
9. BPM, Object Management Group, **Business Process Management Initiative**, 2007. http://www.bpmn.org/
10. B. Bounabat, **Méthode d'analyse et de conception orientée objet décisionnel. Application aux langages synchrones et aux systèmes répartis**, doctoraldiss, Cadi Ayyad University, Faculty of sciences, Marrakech, Morocco, 2000.
11. A Aaroud, S. E. Labhalla, and B. Bounabat, **Modelling the handover function of global system for mobile communication**, The International Journal of Modelling and Simulation, ACTA Press, vol 25, n. 2, 2005. https://doi.org/10.2316/Journal.205.2005.2.205-4136
12. M. Berrada, B. Bounabat, and M. Harti, **Modeling and simulation of Multi-Agent reactif decisionnal systems using business process management concepts**, International Review on Computers and Software (IRECOS), vol. 2, n. 2, pp. 159-169, March 2007.