International Journal of Software Engineering and Its Applications
Vol.12, No.3 (2018), pp.1-18
http://dx.doi.org/10.21742/ijseia.2018.12.3.01

Automatic Code Generation for Web Services Composition Based on a Formal Specification

Meftah Mohammed Charaf Eddine[1] and Kazar Okba[2]

[1]University of Echahid Hamma Lakhdar-El Oued, Algeria
[2]University of Mohamed Khider, BISKRA, Algeria
Mmce2011@yahoo.fr, Kazarokba@yahoo.fr

Abstract

The web service oriented architecture (WSOA) design the software as services and uses the services as platforms. The orchestration of the services describes how Web services can interact with each other in an operational perspective, with control structures, including the order execution interactions. Many languages allow expressing executable processes to implementing Web services orchestration. These languages are used to describe how interactions between multiple services are coordinated to achieve a goal. However, the operational semantics of each of the structures of these languages is not formally defined and their limitations to the reasoning and verification of Web services compositions. This approach involves the automatic code generation for web services composition from formal specifications. We described the application using graphical notations (UML). Next, B automatic process of refinement can be applied to build a web services composition that satisfies the purpose of the web application. Due to the generic nature of the refinement rules, an automatic code generation tool (UML-B-BPEL4SW) can be achieved, thereby reducing the cost of the development. A case study implementation of the proposed approach is developed using B tools.

Keywords: *Web Service, UML, Formal Method B, Refinement Process, BPEL4SW*

1. Introduction

In recent years, the security in the development of Web applications becomes a need and goal significant and crucial.

The Web (2+) technologies such as SOA, design the software as services. More importantly, they are rethinking the services as platforms, rather than viewing services as products; these sets of technologies considered a service is primarily a brick whose role is allow to other services to build other services by using it. In such applications, the difficulty for designers' lies in writing programs that respect the integrity of all transactions and web services orchestration.

The aim of this paper is the definition of a formal approach to developing secure Web services compositions. In these compositions, designers tend to use methods such semiformal say: UML, CTT, OMT ... mainly based on graphical notations (classes, entities, states / transitions ...) or symbolic (as CTT) for intuitive representation, simplified and synthetic system to study. These methods represent undeniable advantages for modelling. They are an ideal medium for communication between the different actors of the system. Nevertheless, these methods still suffer from a lack of precise semantics of their concepts. This lack of semantic greatly reduces the possibility of formal reasoning and evidence obtained on modelling.

Our work depends on the generation of an implementation (BPEL4SW) from formal specifications B (mathematical notations). This generation is performed by successive

Received (April 25, 2018), Review Result (June 28, 2018), Accepted (July 3, 2018)

refinements of abstract specifications obtained by the translation phase of graphical notations. A set of generic rules refinement B has been defined. These rules consider the dynamic aspect (operations) of the web application. The objective of such a translation is obtaining the design model precise, concise and analyzable by tools (Control, evidence, simulation ...).

The rest of the article is organized as follows: Section 2 presents briefly the formal method B, then in section 3 presents briefly BPEL4SW language. Sections 4, presents the related works; next the general approach proposed is described in Section 5; in Section 6 we detail the approach through a case study. A synthetic study of work similar to our approach is given in Section 7. Section 8 concludes and describes potential prospects of our work.

2. Presentation of B Method

B Method [1] is first and foremost a new approach for the specification and design of software that ensures its safety and reliability. All of the specification, design and coding processes are therefore fully based on the realization of a certain number of mathematical proofs. It is only once a model has been mathematically proven that it is considered coherent and fault-free. The main objectives of this method are therefore:

- To create correct software by construction
- To model systems in their environment
- To formalize specifications
- To simplify programming

B is a formal specification method which, thanks to an adequate language, allows for highly accurate expressions of the properties required by specifications. One can then prove in a fully automated fashion that these properties are unambiguous, coherent and are not contradictory. This then allows us to mathematically prove that these properties are taken into account as the design stages progress.

Therefore, this method and its associated proof allow for:

- Clear technical specifications and system specifications to be reached that are structured, coherent and unambiguous,
- The development of software that is contractually guaranteed to be fault-free,

B Method usually refers to the set that includes: B language, refinement, proof and related tools. B development starts with the writing of a concrete model that includes all of the defined needs. The main data processed by the system is described, as well as the fundamental properties of this data. Services ensure the processing of this data while preserving its properties. The B model thus obtained is a specification of what the system should create.

B model is then transformed (refined with B vocabulary) until a complete software installation of the software is obtained. B Method is therefore: “a proven construction approach (referred to as correct) based on B Language, refinement and proof.” [2]

The B language basics are the *abstract machine* and *refinement:*

An abstract machine describes a system through a set of variables called states. Typing and constraints governing these variables, writings in a variation of the set theory Zermalo- Frankel (ZF), are expressed in the INVARIANT clause machine.

MODEL nameM

REFINES nameR

SETS . . .

PROPERTIES . . .

VARIABLES . . .

INVARIANT . . .

ASSERTIONS . . .

INITIALISATION . . .

EVENTS . . .

END

The dynamic part of the system is specified by a set of operations B. The operations are described in the language of Disjkstra substitutions:
Generalized substitutions:

[SKIP]P ⇔ *P*

[S1 II S2]P ⇔ *[S1]P ^ [S2]P*

[ANY v WHERE E THEN S END]P ⇔ *¥ v.(E => [S]P)*

[SELECT E THEN S END]P ⇔ *E => [S]P*

[BEGIN S END]P ⇔ *[S]P*

[x := E]P ⇔ *P(x/E)*

The basic substitution is assigning a value to a variable. The generalization of this language allows defining more sophisticated substitutions: non-deterministic replacing preconditioned substitution....

A preconditioned substitution is of the form:

SELECT *p* **THEN** *s* **END**,

p is a predicate and s is a substitution. If *p* checked the substitution s is executed correctly; if not s fails. The result of the execution of the substitution s is unpredictable.
Method B offers several architectural clauses allowing to incremental specification. At the most abstract level, the most used link INCLUDES. A machine may be included more than once in a machine B: variable A can be read from B, the update of these variables is only possible through the use of operations of A. This restriction allows a separation of proof associated with two machines.
Indeed, at B, each refinement step is validated by the establishment of a set of proof obligations generated automatically, which guarantees the correction of various transformations wrought by the refinement.

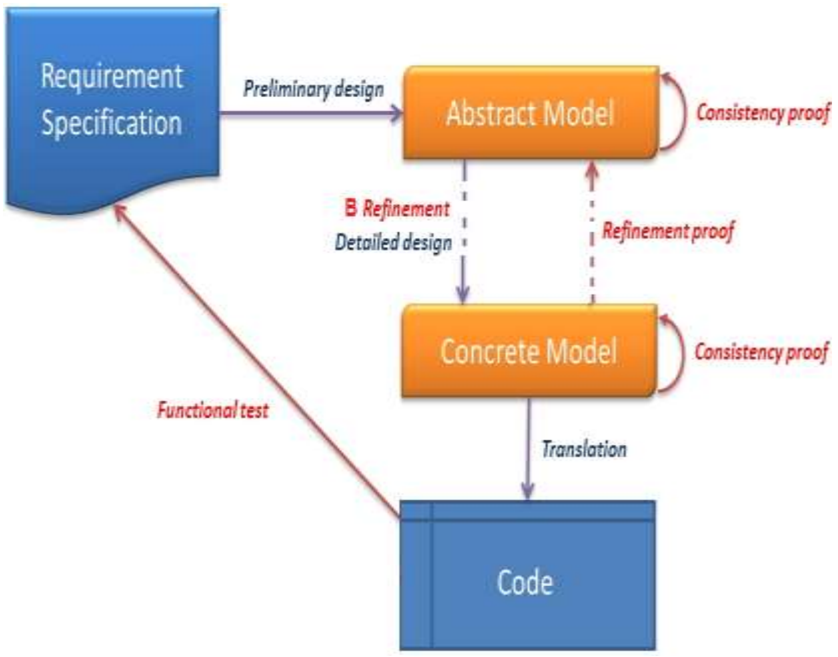


Figure 1. Structuring of a Developing B [2]

However, in general, the refining process is still a manual task, relatively heavy, particularly in proof phase. Method B is implemented by various tools including Click'n'Prove / B4Free v2 [2].

3. Business Process Execution Language for Web Services (bpel4sw) Language

Business Process Execution Language (BPEL) [3] defines a notation for specifying business process behavior based on Web Services. Business processes can be described in two ways:

- Executable business processes model actual behaviour of a participant in a business interaction.
- Business protocols, in contrast, use process descriptions that specify the mutually visible message exchange behaviour of each of the parties involved in the protocol, without revealing their internal behaviour. The process descriptions for business protocols are called abstract processes.

BPEL is used to model the behavior of both executable and abstract processes. The scope includes:

- Sequencing of process activities, especially Web Service interactions
- Correlation of messages and process instances
- Recovery behavior in case of failures and exceptional conditions
- Bilateral Web Service based relationships between process roles [3].

Processes in BPEL export and import information by using web service interfaces. BPEL describe Web-service interactions, this interaction is ensured through Partnerlink. They represent the static part of a BPEL process and are described in a WSDL document.
The dynamic part of BPEL is described by two types of activities:

A. Basic activities: define basic operations of a business process as:

- <*invoke*> to invoke an operation in a service.
- <*Receive*>, wait a message from an external source.
- <*reply*>, in response to an external source.
- <*wait*>, wait for a period of time.
- <*assign*>, to copy data from one place to another.
- <*throw*> to launch a runtime error.
- <*terminate*>, to end the instance of service.
- <*empty*>, which does nothing (useful for synchronizing parallel activities).

B. Structural activities: define the order in which the nested activities are executed, such as:

- <*sequence*>, for sequential execution order.
- <*flow*>, for parallel execution.
- <*switch*>, for conditional execution routing.
- <*while*>, for loops.
- <*pick*> to wait an event.
- <*scope*>, activity allows decomposing the BPEL process into sub-processes.

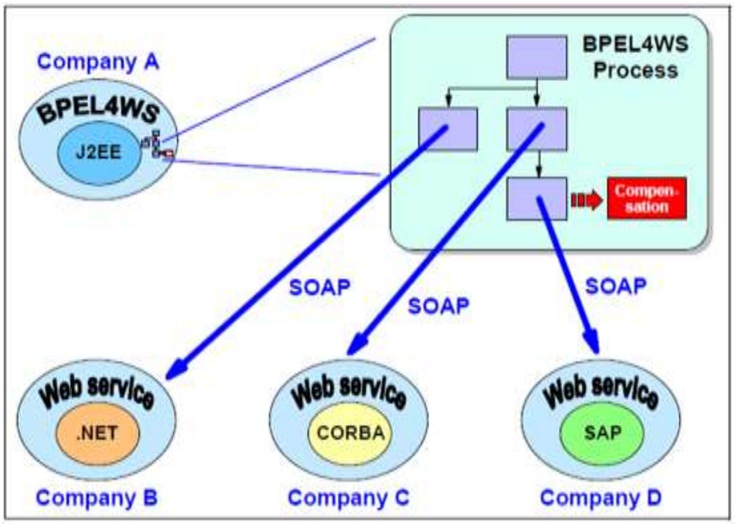


Figure 2. Web Services Orchestration (BPEL4SW)

4. Related Work

To overcome the lack of formality in this area, several approaches have been proposed to this effect, based on the work related to transition systems [4-7], the process algebra [8] [9] or temporal theories [10-11]. These works are devoted to the formalization of Web services orchestrations and allow some checks of their behavior. This verification step will help to ensure a certain level of confidence in the internal behavior of an orchestration. It may be noted that all of this work includes both the expression of an executable process, the expression properties, and validation of this process with regard to the different properties.

However, these proposals works are partial solutions to the problem of development of secure Web services composition. The specifications generated are too abstract to be directly supported by an implementation language. These specifications correspond to the conceptual level.

A refinement stage (coding) of these specifications is essential.

5. The Proposed Approach

An important concept in the design of the Web services composition is the task. These tasks describe the various interactions between web services (orchestration) to achieve a particular goal.

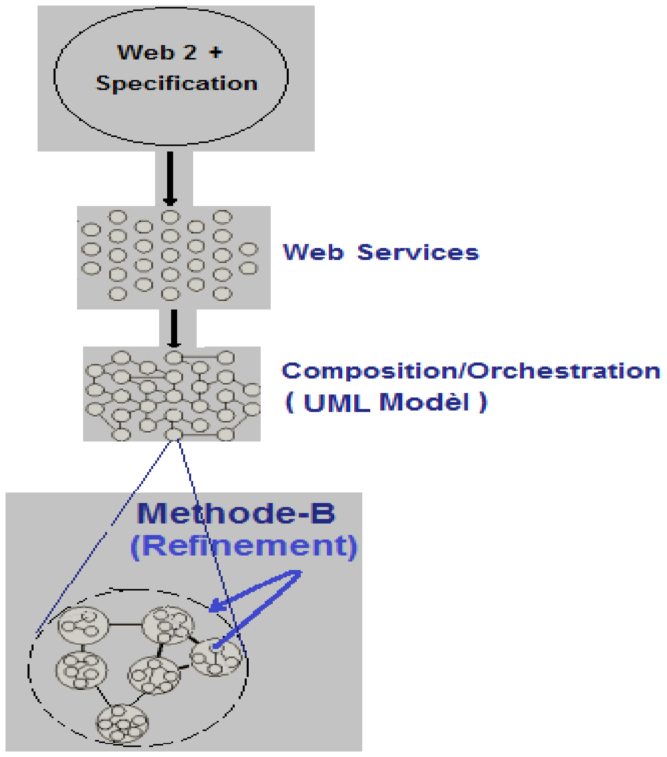


Figure 3. Tools used (UML and B)

5.1. General Architecture of the Proposed Approach

The activity model that we have considered is described by the UML activity diagram.

The design of an orchestration calls particularly to descriptions of activities of Web services to describe the actions of task that a services are achieved using this orchestration.

Web services compositions (orchestration) can be specified by the composition of elementary tasks with the operator UML activity diagram. We represent the composition of web services using the refinement. This approach (Figure 4) allows the inclusion of UML activity models in the developments in the B event.

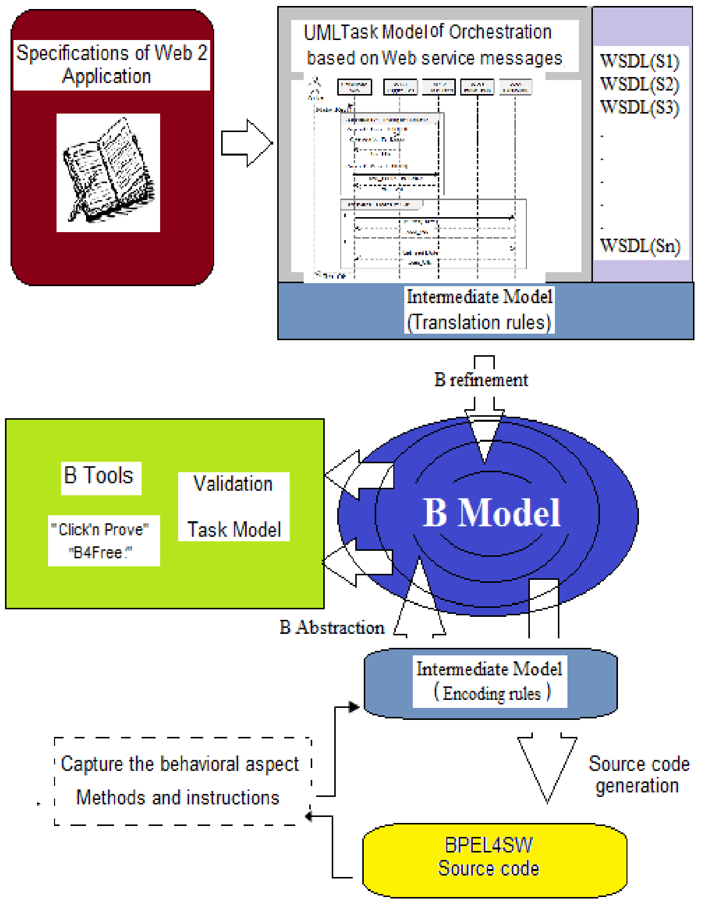

Specifications of Web 2 Application
UMLTask Model of Orchestration based on Web service messages
WSDL(S1)
WSDL(S2)
WSDL(S3)
WSDL(Sn)
Intermediate Model (Translation rules)
B refinement
B Tools
"Click'n Prove" "B4Free"
Validation
Task Model
B Model
B Abstraction
Intermediate Model (Encoding rules)
Capture the behavioral aspect
Methods and instructions
Source code generation
BPEL4SW Source code

Figure 4. The Proposed Approach

Regarding the verification, two approaches can be differentiated:

- The first is translating the source code of application into a formal language in order to check as in the following Figure (4):

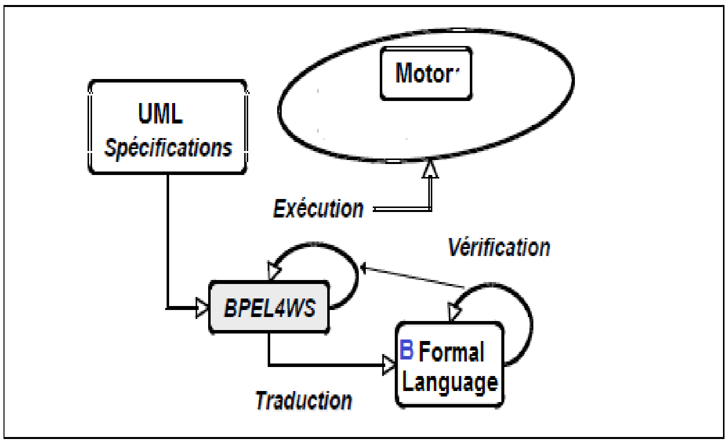


Figure 5. From BPEL4SW to Language B (ascending)

The second is to express a behavioural aspect of the process with a formal language, in order to check, and then translate it into source code, as in Figure 5:

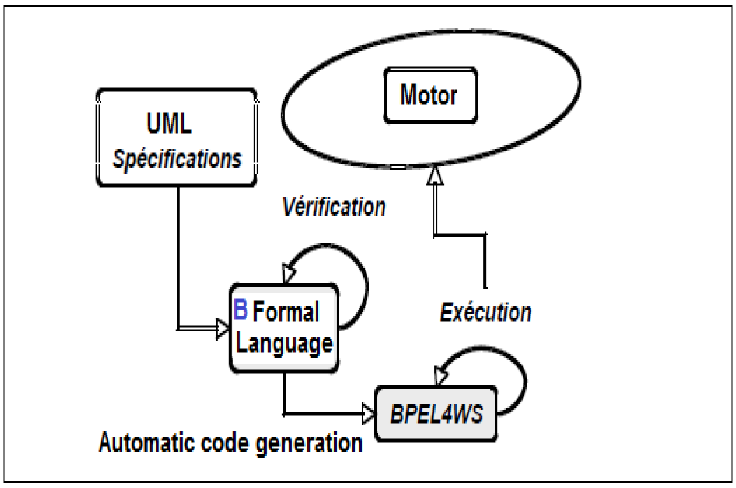


Figure 6. From BPEL4SW to Language B (descending)

5.2. The Refinement of a Composition

Refinement is the process of transforming abstract specifications composition of web services to more concrete specifications. Generally, refinement step is the most critical and difficult phase of B project development. There is no precise method that would refine any abstract specification to a language. Indeed, such a method should be able, for each possible implementation, determining concrete variables and also the linking invariants binder these concrete variables to abstract variables of the specification. However, the definition of a special refinement process for a specific area is possible.

In our field (web services) the abstract specification of web services composition it's the specification of the nature of the composition between services, it can take two forms:

Order: a composition can execute the services components sequentially or in parallel.

Alternative: a composition may invoke alternative services until one of them successful.

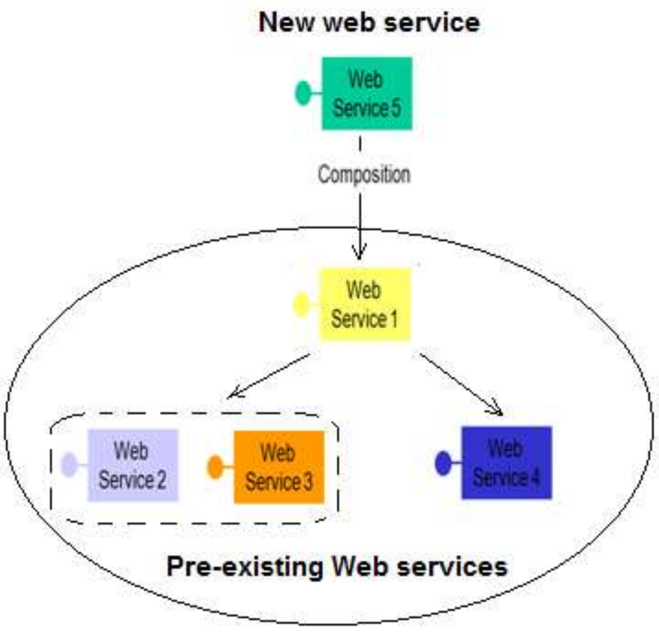


Figure 7. Web Services Composition

In B, there are two types of refinement:

Data refinement: it is the replacement of an abstract data *D* by concrete data *D'*. In this case, a predicate J, called invariant bonding. This invariant establishes the link between the data D and D '. The invariant bonding J is specified in the INVARIANT clause.

In our field (web services) invariant bonding establishes the link between WSDL data (abstract Part: Types, messages, portType and concrete part: binding, service) of the services composed with the same WSDL date of the services components.

Algorithmic refinement: it is the transformation of an abstract substitution S in a less abstract substitution S' (*e.g.*, replacement of a simultaneous substitution by sequential substitution and elimination of preconditions).

In our field (web services) applying this type of refinement on basic operations of the services activities; we consider that each service as a set of operations interact with a set of messages.

5.3. Dependence of Messages

Dependence message defines the mapping of inputs and outputs messages. There are three types of dependence:

Synthesis: This type combines the output messages of the components services to form the output messages of the composed service.

Decomposition: This type decomposes the input message of the composed service to generate the input messages of components services.

Correspondence messages: allows the correspondence between the inputs and outputs messages of the services.

These two types of refinement (data and algorithmic) are not mutually exclusive: they can be operated in the same stage of refinement. Obviously any data refinement leads to an algorithmic refinement.

The last level of refinement of an abstract machine is called implementation (described in a language called B0). The language describing this implementation uses data structures and programs supported by the programming language chosen (BPEL4SW). The translation of this implementation (B0) to the BPEL4SW language becomes very natural and it can be automatically.

5.4. Translation Rules

B (Descending / Ascending) translation requires translation rules to represent the control structures:

A. Descending translation (UML-B)

Capturing the behavior aspect (activity diagrams) of the composition process. Then represent the execution of services activities as a set of processes (sequential or parallel include assignments) in B. These assignments are scheduled by a set of control structures of B : *sequential composition, loops, if then else ...*

B. Ascending translation (BPEL4SW-B)

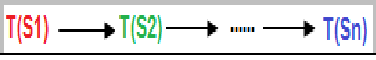
Capturing the behavioral aspect of source code (BPEL4SW). Then represent the execution of these instructions as a set of processes: sequential or parallel assignments of B.

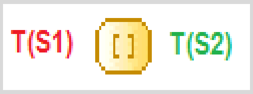
C. Encoding Rules (B-BPEL4SW):

The abstract specifications (B), are refined until the level of abstraction (language B) corresponding to the formalization of the concepts of source code BPEL4SW. This level of abstraction is defined as a set of assignments. These assignments are scheduled by a set of control structures BPEL4SW: sequential composition loops, if then else...

The abstract model of Web services composition (orchestration process) consists of the main events are: Invoke Service 1 and Invoke Service 2, ..., Invoke service n whose role is to invoke an operation in a partner services (Services 1 Service 2 ... Service n). It can be used for both synchronous and asynchronous communication. These events are synchronized by variables: invokeS1, invokeS1 ... invokeSn (which have a boolean behavior (1/0) to indicate that the service is called or not); and S1pick, S2pick, ..., Snpick variables (which have a Boolean behavior (1/0) to indicate that the service is in a state of waiting the arrival of an event or not).

Table 1. Translation Rules (UML-B-BPEL4SW)

UML Specifications (semi-formal)	***Sequential composition:*** 
Model B **(Formal)** + **(Refinement)** + **(Proof)**	**VARIABLES** ***Invoke Service 1, Invoke Service 2,….., Invoke Service n*** **INVARIANT** ***invokeS1 ϵ (0,1)∧ invokeS2 ϵ (0,1) ∧…∧ invokeSn ϵ (0,1)*** ***S1 pickϵ (0,1)∧ S2 pick ϵ (0,1) ∧…∧ Sn pickϵ (0,1)*** ***invokeS1≠ invokeS2≠……≠ invokeSn*** ***S1 pick ≠S2 pick ≠…≠ Sn pick*** **EVENTS** **...** ***Invoke Service 1*** **= SELECT** ***invokeS1= 1^ S1pick=1*** ***…*** **THEN** ***invokeS1:= 0 ^ S1pick:= 0 II…*** ***invokeS2 := 1 II…*** ***…*** ***END;*** ***Invoke Service 2*** **= SELECT** ***invokeS2= 1^ S2pick=1*** ***..*** ***THEN*** ***invokeS2:= 0 ^ S2pick:= 0 II…*** ***invokeS3:= 1 II…*** ***…*** ***END;*** **…….** ***Invoke Service n = SELECT*** ***invoke S n = 1^ S1pick=1*** ***..*** ***THEN*** ***invoke S n := 0 ^ Snpick:= 0 II…*** ***…*** ***END;*** **END;**
Source code (BPEL	

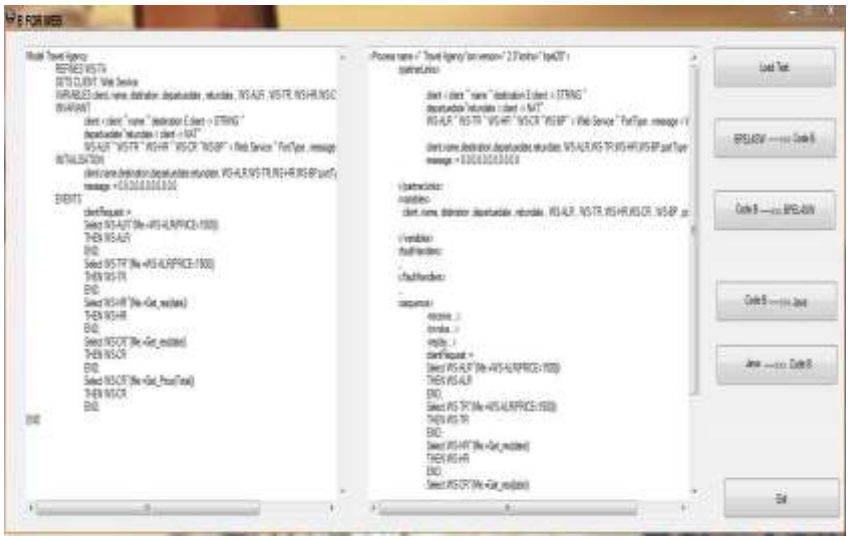
4SW)	**<sequence>** *S1* *S2* . . . *Sn* **<sequence>**
UML Specifications (semi-formal)	***Conditional compositions:*** 
Model B **(Formal)** + **(Refinement)** + **(Proof)**	**VARIABLES** ***Invoke Service 1, Invoke Service 2*** **INVARIANT** ***invokeS1 ϵ (0,1)^ invokeS2 ϵ (0,1)*** ***S1 pickϵ (0,1)^ S2 pick ϵ (0,1)*** ***Cond ϵ (0,1)*** ***invokeS1≠ invokeS2*** ***S1 pick ≠S2 pick*** **EVENTS** **...** ***Invoke Service 1* = SELECT** ***invokeS1= 1^ S1pick=1^ Cond=1*** ***…*** **THEN** ***invokeS1:= 0 ^ S1pick:= 0 II…*** ***…*** ***END;*** ***Invoke Service 2* = SELECT** ***invokeS2= 1^ S2pick=1^ Cond=0*** ***…*** **THEN** ***invokeS2:= 0 ^ S2pick:= 0 II…*** ***…*** ***END;*** **…….** **END;**

Source code (BPEL 4SW)	**if Cond then T(S1)**
	else *T(S2)*

UML Specifications (semi-formal)	***Iterative loops composition:***
	VARIABLES

Model B **(Formal)** + **(Refinement)** + **(Proof)**	***Invoke Service 1*** **INVARIANT** ***invokeS1 ϵ (0,1)*** ***S1 pickϵ (0,1)*** ***Condi ϵ (0,1)*** **EVENTS** **...** ***Invoke Service 1* = SELECT** ***invokeS1= 1^ S1pick=1^ Condi=1*** ***…*** **THEN** ***invokeS1:= 1 ^ S1pick:= 1 II…*** ***…*** ***END;*** **…….** **END;**
Source code (BPEL 4SW)	***While Condi {T(S1)}***

Based on the previous rules (Table 1) we have developed [14] a code transformation tool (B-BEL4SW) and (BPEL4SW-B):



```
Model Travel Agency
    REFINES WS-TA
    SETS CLIENT, Web Service
    VARIABLES client, name, destination, departuredate, returndate, WS-ALR, WS-TR, WS-HR, WS-C[illegible]
    INVARIANT
        [illegible]
    INITIALISATION
        [illegible]
    EVENTS
        clientRequest ≙
        Select WS-ALR [illegible]
        THEN WS-ALR
        END
        Select WS-TR [illegible]
        THEN WS-TR
        END
        Select WS-HR [illegible]
        THEN WS-HR
        END
        Select WS-CR [illegible]
        THEN WS-CR
        END
        Select WS-CR [illegible]
        THEN WS-CR
        END
END
```

```
<Process name = "Travel Agency" [illegible]>
    <partnerLinks>
        [illegible]
    </partnerLinks>
    <variables>
        [illegible]
    </variables>
    <faultHandlers>
    </faultHandlers>
    <sequence>
        <receive [illegible]
        <invoke [illegible]
        <reply [illegible]
        clientRequest ≙
        Select WS-ALR [illegible]
        THEN WS-ALR
        END
        Select WS-TR [illegible]
        THEN WS-TR
        END
        Select WS-HR [illegible]
        THEN WS-HR
        END
        Select WS-CR [illegible]
```

Figure 8. A Code Conversion Tool (B-BPEL4SW, BPEL4SW-B)

6. Case Study: Build the Complete Application (travel agency)

A travel agency (service01) provides the following services: reservation airline tickets, reservation hotel rooms and car rental. in order to providing these services to its clients, the agency should establish links with other services: airlines (service02), hotels (service03), car rental companies(service04) and banks (service05) to facilitate financial transactions between clients and the travel agency and between the agency and other partners. For this purpose, five web services can be offered:

- WS-AV: provides an interface with its clients, and establish links with other services,

- WS-VOL: used to flights reservation according to a specified date, the city of departure and arrival and number of people.
- WS-Hotel: used to hotel rooms reservation according to the date and the number of people.
- WS-LV: used to reserve cars by a specified date and number of days.
- WS-Bank: used to pay different reservations.

The following figure shows the services of our application:

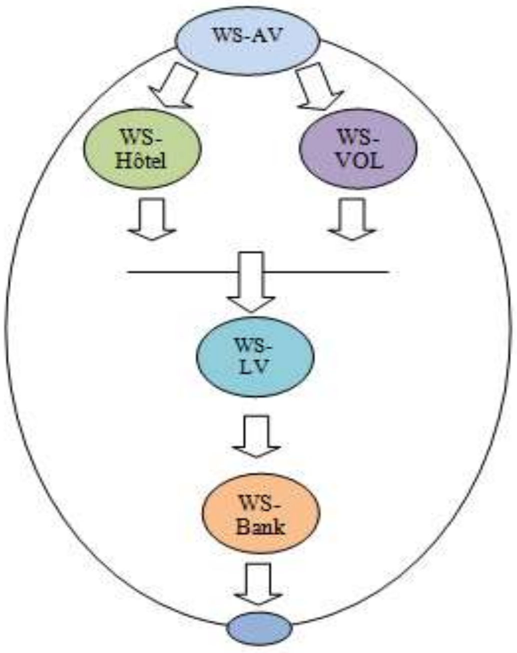


Figure 9. Web Services of a Travel Agency

In this example the service (WS-AV) is the composition of four services (WS-VOL, WS-Hotel, WS-LV and WS -Bank).

The four services: Airline, Hotel, Car rent and Bank Web services have been exposed over the network and their WSDL file can be accessed by URLs as following:

Table 2. WSDL URLs for Web Services of a Travel Agency

Web Services	URL
WS-VOL	http://localhost:8080/AirlineReservation/AilrlineReservationService?WSDL
WS-Hotel	http://localhost:8080/HotelReservation/HotelReservationService?WSDL
WS-LV	http://localhost:8080/CarrentReservation/CarrentReservationService?WSDL
WS -Bank	http://localhost:8080/BankPayer/BankPayerService?WSDL

Netbeans tool enables also to import the WSDL files for any external web service provider. Thus, we import the five WSDL files for the five service providers with their xml Schema. WSDL of the process imports the WSDL of other web services providers and defines the partnerlink types for them.

The five partner link types are as following:

- ClientPartnerLink: used to describe the interaction between the client and the BPEL process itself. This interaction is synchronous. This partner link type is declared in the WSDL of the BPEL process.
- AirlinePartnerLink: used to describe the interaction between the Airline service provider and the BPEL process itself. This interaction is synchronous. This partner link type is defined in the BPEL process.
- HotelPartnerLink: used to describe the interaction between the BPEL process and the Hotel Reservation Web service. This interaction is synchronous. This partner link type is defined in the BPEL process.
- CarPartnerLink: describes the interaction between the BPEL process and the Car rent Web service. This interaction is synchronous. This partner link type is defined in the BPEL process.
- BankPartnerLink : describes the interaction between the BPEL process and the Bank Web service. This interaction is synchronous. This partner link type is defined in the BPEL process.

6.1 Application Development

The role of WS-AV is to organize the trip through the internet. The service provides to clients an interface to input the following information:

Departure date, return date, departure city, arrival city, number of persons and return the tickets and the total price of the trip.

The internal scenario that will lead the interactions between services is as follows: The WS-AV service will first connect both the WS-hotel services and WS-VOL through its methods reserveCHam respectively () and reserveVol (), later and after thes previous reservations, it will connect to the WS-LV Service by reserveVoiture () method. then it will send the price of each service to WS-Bank service to calculate the total price and paid by his method liabilities () as shown in the following UML activity diagram.

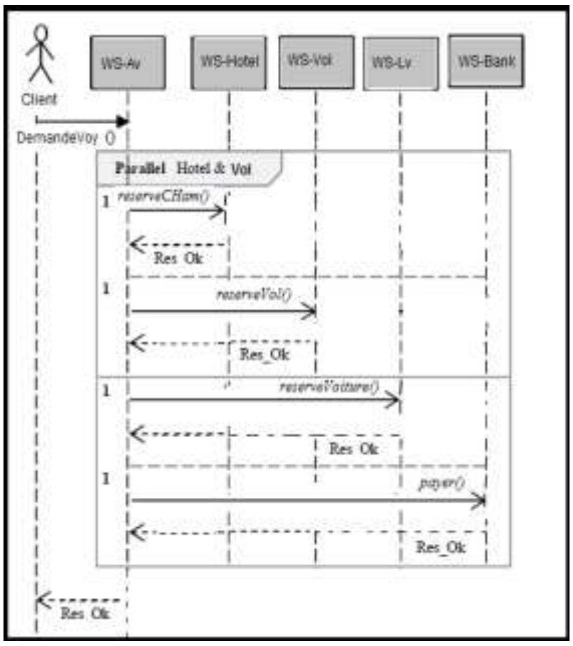


Figure 10. UML Activity Diagram of a Travel Agency

The application functionalities are described using UML activity diagrams. A precise semantics of these diagrams was defined in [13]. These diagrams are the entrance of the translation process by B. This is described by a set of formal rules defined in (V.C). The development of an orchestration will be by coding the expression of activity by refinements B using the operators' refinement patterns.

The specifications generated during the translation stage can be divided into several abstract machines (B) interconnected by the INCLUDES clause.

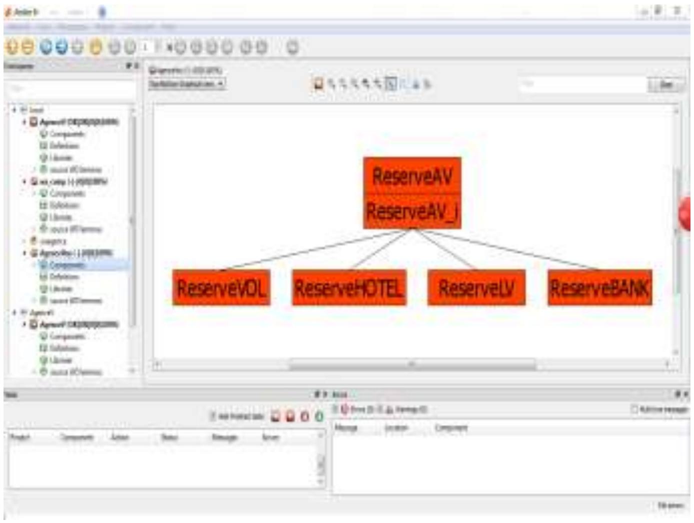
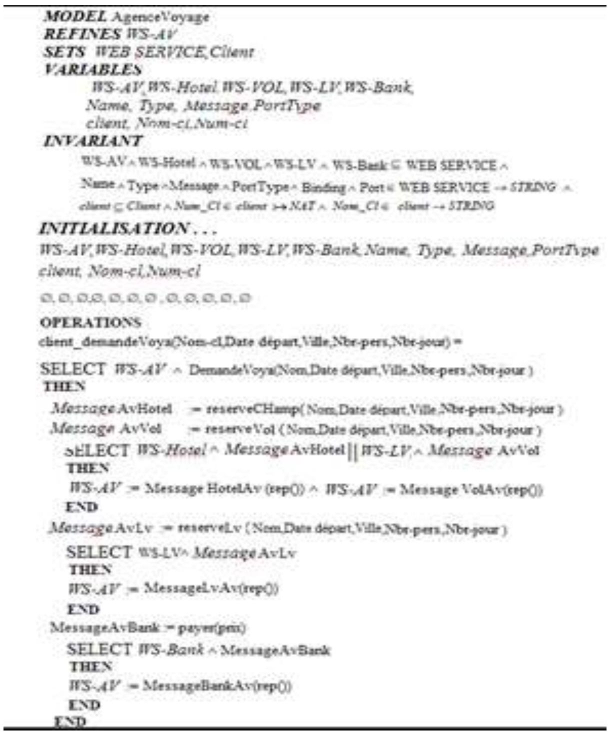


Figure 11. Model B of Travel Agency Web Service

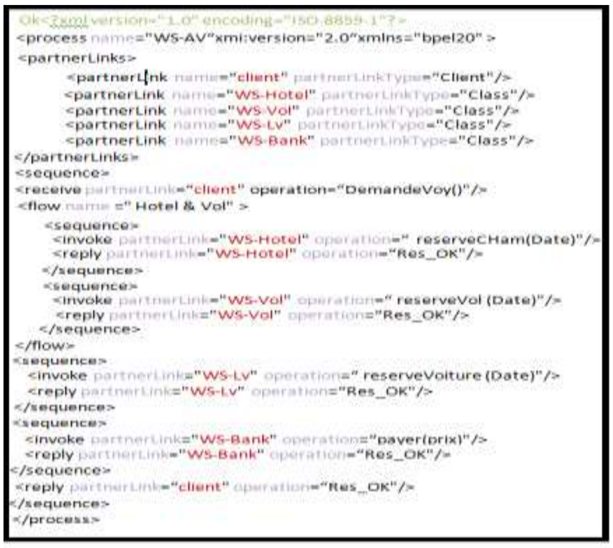
The objective of such modularization is to reduce the size and complexity of the machines, thus improving the readability of the specifications and elucidate the proof phase.



```
MODEL AgenceVoyage
REFINES WS-AV
SETS  WEB SERVICE,Client
VARIABLES
      WS-AV,WS-Hotel WS-VOL, WS-LV, WS-Bank,
     Name, Type, Message PortType
     client, Nom-cl,Num-cl
INVARIANT
     WS-AV ∧ WS-Hotel ∧ WS-VOL ∧ WS-LV ∧ WS-Bank ⊆ WEB SERVICE ∧
     Name ∧ Type ∧ Message ∧ PortType ∧ Binding ∧ Port ∈ WEB SERVICE → STRING ∧
     client ⊆ Client ∧ Num_Cl ∈ client ↣ NAT ∧ Nom_Cl ∈ client → STRING
INITIALISATION . . .
WS-AV,WS-Hotel,WS-VOL,WS-LV,WS-Bank,Name, Type, Message,PortType
client, Nom-cl,Num-cl
∅,∅,∅,∅,∅,∅,∅,∅,∅,∅,∅,∅,∅
OPERATIONS
client_demandeVoya(Nom-cl,Date départ,Ville,Nbr-pers,Nbr-jour) =
SELECT WS-AV ∧ DemandeVoya(Nom,Date départ,Ville,Nbr-pers,Nbr-jour )
 THEN
  MessageAvHotel    := reserveCHamp( Nom,Date départ,Ville,Nbr-pers,Nbr-jour )
  Message AvVol     := reserveVol ( Nom,Date départ,Ville,Nbr-pers,Nbr-jour )
   SELECT WS-Hotel ∧ MessageAvHotel || WS-LV ∧ Message AvVol
   THEN
   WS-AV := Message HotelAv (rep()) ∧ WS-AV := Message VolAv(rep())
   END
  MessageAvLv := reserveLv ( Nom,Date départ,Ville,Nbr-pers,Nbr-jour )
   SELECT WS-LV∧ MessageAvLv
   THEN
   WS-AV := MessageLvAv(rep())
   END
  MessageAvBank := payer(prix)
   SELECT WS-Bank ∧ MessageAvBank
   THEN
   WS-AV := MessageBankAv(rep())
   END
  END
```

Figure 12. Machine B for WS-AV

The abstract specifications, generated in the previous step, are refined until the level of abstraction (language B0) corresponding to the formalization of the concepts of source code BPEL4SW. A set of generic rules of basic instructions is defined in paragraph (5.4).



```
Ok<?xml version="1.0" encoding="ISO-8859-1"?>
<process name="WS-AV"xmi:version="2.0"xmlns="bpel20" >
<partnerLinks>
        <partnerLink name="client" partnerLinkType="Client"/>
        <partnerLink name="WS-Hotel" partnerLinkType="Class"/>
        <partnerLink name="WS-Vol" partnerLinkType="Class"/>
        <partnerLink name="WS-Lv" partnerLinkType="Class"/>
        <partnerLink name="WS-Bank" partnerLinkType="Class"/>
</partnerLinks>
<sequence>
<receive partnerLink="client" operation="DemandeVoy()"/>
<flow name =" Hotel & Vol" >
    <sequence>
     <invoke partnerLink="WS-Hotel" operation=" reserveCHam(Date)"/>
     <reply partnerLink="WS-Hotel" operation="Res_OK"/>
    </sequence>
    <sequence>
     <invoke partnerLink="WS-Vol" operation=" reserveVol (Date)"/>
     <reply partnerLink="WS-Vol" operation="Res_OK"/>
    </sequence>
</flow>
<sequence>
  <invoke partnerLink="WS-Lv" operation=" reserveVoiture (Date)"/>
  <reply partnerLink="WS-Lv" operation="Res_OK"/>
</sequence>
<sequence>
  <invoke partnerLink="WS-Bank" operation="paver(prix)"/>
  <reply partnerLink="WS-Bank" operation="Res_OK"/>
</sequence>
<reply partnerLink="client" operation="Res_OK"/>
</sequence>
</process>
```

Figure 13. BPEL 4SW Code for WS-AV

7. Comparison with Existing Works

In this section we propose a model for comparison between works and approaches in this field. The proposed model adopts the concepts: formalization, aspect and automation of development processes. The comparison shows the importance of the contribution presented in this paper.

Table 3. Comparison with Existing Works

Approaches	Formalization	Aspect	Code generation
Our approach	Formal	Behavior	Automatic (BPEL4SW)
Transition systems LTSA-WS [4] [6] [7]	Formal	Functional	not automatic
Petri net [5].	Semi-formal	Behavior	not automatic
L- process algebra-ASDL[8] [9]	Formal	Functional	not automatic
LOTOS/CADP[10] [11]	Formal	Functional	not automatic
Diapason [13].	Formal	Behavior	not automatic
UML-Colombo-WS [15].	Formal	Behavior	automatic (XML)
Le π -calcul [16] [17]	Formal	Behavior	not automatic
MDA-UML-S [18]	Semi-formal	Behavior	not automatic

8. Conclusion and Perspectives

This work provides a solution for the formal modeling and automatic generation of the composition of web services. Indeed, many proposed approaches to formalizing and modeling of web services, however, these approaches are partial solutions to the problem of development of safe composition. The specifications generated are too abstract to be directly supported by an implementation language. These specifications correspond to the conceptual level considered development. A refinement stage (coding) of these specifications is essential.

Our approach is based primarily on the refinement B. The process described in this paper is dedicated to the generation of a BPEL4SW implementation from a semi-formal specification that reflects the activities of a web application described using UML diagrams. We have defined a set of rules which allows not only the refinement of the data but also the associated operations.
The approach presented in this article has several advantages:

• Reduced costs of development: automation phases specification, refinement, and automatic code generation

• Standardization of the generated code: the two phases of translation and refinement dictated by precise and deterministic rules. These code standards provide a better understanding and code maintenance and product.

• The proposed approach is a top-down / bottom-up approach to formal specification, the formal validation and automatic code generation orchestrations (BPEL4SW).

Our current research focuses on:

- The realization of an automatic refinement tool. This tool will be complement this tool, which allows the automatic generation of WSDL documents related to composed services.
- The realization of an automatic translation tool. This tool will complement this tool, allowing automatic translation of UML diagrams and specifications B.

So these tools, will aim to assist the designer during the development process of its web applications. Such a tool will relieve the designer of different manual and most costly phases of the development process.

We also work on the same ideas to apply them to another technology (AJAX) for aspect (dynamic web interfaces) of Web 2+ applications.

References

[1] Abrial. J. R The B-Book, Combridge University Press, **(1996)**.
[2] B. Method Website: http://www.methode-b.com/en/b-method/ (Clearsy System Engineering)
[3] BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems: “Business Process Execution Language for Web Services Specification”, version 1.1 dated **(2003)** May 5.
[4] F. Andrews, H. Curbera, Y. Dholakia, J. Klein Goland, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic and S. Weerawarana, “Business process execution language for web services version 1.1”, http://www-128.ibm.com/developerworks/library/specification/ws-bpel, **(2003)**.
[5] Hamadi and B. Benatallah, “A petri net-based model for web service composition”, Fourteenth Australasian Database Conference (ADC2003), **(2003)**.
[6] S. Uchitel Foster, J. Magee and J. Kramer, “Model-based verification of web service compositions”, IEEE Automated Software Engineering (ASE), **(2003)**.
[7] S. Foster, J. Magee and J. Kramer, “Tool support for model-based engineering of web service compositions”, IEEE International Conference on Web Services (ICWS), (**2005)**.

[8] S. Uchitel Foster, J. Magee and J. Kramer, “Ltsa-ws: A tool for model-based verifi-cation of web service compositions and choreography”, IEEE International Conference on Software Engineering (ICSE 2006), **(2006)**.
[9] A. Ferrara Salaun and A. Chirichiello, “Negotiation among web services using lotos/cadp”, European Conference on Web Services (ECOWS 04), **(2004)**.
[10] Chirichiello and G. Salaun, “Encoding abstract descriptions into executable web services: Towards a formal development negotiation among web services using lotos/cadp”, IEEE/WIC/ACM International Conference on Web Intelligence (WI 2005), **(2005)**.
[11] A. Cau Solanki and H. Zedan, “Asdl : A wide spectrum language for designing web services”, 15th International World Wide Web Conference (WWW2006), **(2006)**.
[12] W. Gaaloul Rouached, W. M. P. van der Aalst, S. Bhiri and C. Godart, “Web service mining and verification of properties: An approach based on event calculus”, OTM Confederated International Conferences, **(2006)**.
[13] F. Pourraz, “Diapason : une approche formelle et centrée architecture pour la composition évolutive de services Web Préparée au sein du LISTIC : Laboratoire d’Informatique, Systèmes”, Traitement de l’Information et de la Connaissance, **(2007)**.
[14] Meftah Mohammed Charaf Eddine and Hani nabil: Decembre, **(2015)**.
[15] A. Elgammal and M. El-Sharkawi, “Using UML to Model Web Services for Automatic Composition”, Int. J. of Software Engineering, IJSE, vol. 3, no. 2, **(2010)** July.
[16] R. Milner, “Communication and concurrency”, Prentice Hall, **(1989)**.
[17] R. Milner, “Communicating and mobile systems: the _-calculus”, Cambrige University Press, **(1999)**.
[18] C. Dumez Thèse Doctorat : Approche dirigée par les modèles pour la spécification, la vérification formelle et la mise en œuvre de services Web composés (MDA-UML-S) de l’Université de Technologie de Belfort-Montbéliard, **(2010)**.

Authors

Dr. Meftah Mohammed Charaf Eddine, received his master degree in Computer Science from University of Eloued since 2010. Received his PhD degree in Computer Science from University of Biskra since 2016. Currently a lecturer in the Department of Computer science at the EL-Oued University (Algeria). His research interest includes Software engineering, Web services and Formal approach.

Pr. Okba Kazar, received his master degree in 1997 from the Constantine University (Algeria) working on artificial intelligence field. He obtained his PhD degree from the same university in 2005. He is member of editorial board of some Journals. He is an author of some publication in international journals and session chair in international conferences. Actually Okba KAZAR is full professor at computer science department of Biskra University and director of intelligent computer science laboratory. He is interested to the multi-agents systems and their applications, advanced information systems, Web services, semantic Web, bigdata, internet of things and cloud computing.