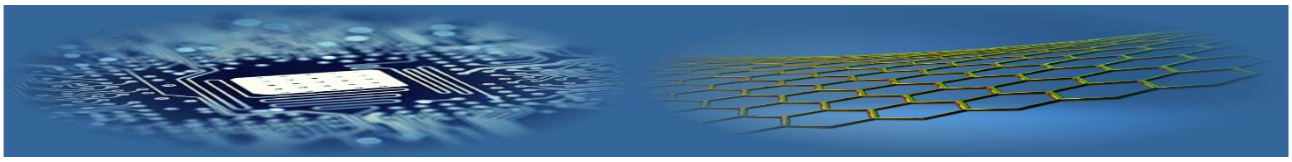

Proposition of web services composition approach basing of model-driven approach and multi-agent systems

N Adadi[1]*, M Berrada[1], D Chenouni[1], B Bounabat[2]

[1]IPI Laboratory, Sidi Mohamed ben Abdellah University, Fez, Morocco
[2]Al-Qualsadi Research & Development Team, ENSIAS, Mohammed V – Souissi University, Rabat, Morocco

**Corresponding author: nouha.adadi@usmba.ac.ma*

Received 23 February 2017, www.cmnt.lv

Abstract

Web services composition is an emerging paradigm for application integration within and across organizations and enterprises. For this reason, various approaches and formalism have been proposed and used for web services composition. Among these approaches we have the Models Driven Approach (MDA), which concentrates on the realization of abstract models. Thus, the phase of specification represents an important part of the cycle of development of composite web service. To proceed to this cycle of development, a developer has to elaborate a specification which allows the modelling of the global behaviour of the system, to verify formally this model for assuring his quality, then pass to the implementation of the composed service. In the paper we present a summary of our proposed approach of web services composition based on MDA, thus it is separated into three tasks: specification using BPMN notation and Multi-agent reactive decisional (MARDS) model, formal verification using LOTOS language and implementation using BPEL language. Then we present a case study to prove the feasibility and reliability of our proposed approach.

Keywords:

Web services composition
MDA; Specification
MARDS
Formal verification

1 Introduction

Nowadays Web Services are defined as software components, which can be invoked by application programs through a stack of Internet standards. Once deployed, web services provided by various organizations can be inter-connected in order to implement business collaborations, leading to composite web services. In the literature several approaches are proposed in order to compose web services, these approaches can be grouped into four classes: workflow-based approaches [1], approaches based on artificial intelligence planning techniques [2], approaches based on dependence graphs [3], and model-driven approaches.

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 Model-driven approach.

The layout of this paper is as follows. In the second section, we present a summary of our proposed approach of web services composition, the third section is devoted to the case study, we present a web services scenario that is used to apply and explain the different steps of our proposed development process. The conclusion and future work are presented in section IV.

2 Proposed approach

In this section, we present a summary of our proposed approach based on MDA and we explain the process of development of composite service. The figure 1 shows the steps involved in the proposed development process (specification, formal verification and implementation) to better understand how to proceed.

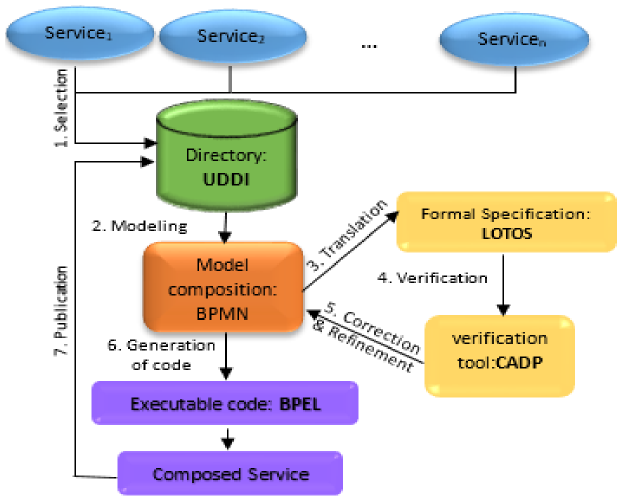

FIGURE 1 Process of development of composite service

2.1 PHASE OF SPECIFICATION

The specification phase is very important because it allows to detach from the implementation to realize clear abstract models, helping to the overall understanding of the system.

Furthermore, this specification is generally sufficiently expressive to serve as a basis for the implementation and even possibly to enable the generation of code in an automated manner.

In the process presented in figure 1, once the requested services are selected by the directory we pass to the specification stage. At this level we propose a modelling based on MARDS (Multi-Agent Reactive Decisional System) model [4], and using the BPMN notation (Business Process Model and Notation) [5]. The MARDS model, constitutes an approach among the newest and most useful ones for the composing and modeling of complex system such as the automated systems of production, the mobile systems [6] and organizational system [4]. We have used this system in our approach because it allows to model the composition of services in a simple and powerful way, and in well-structured architecture. The BPMN notation, is a modeling language, it is more adapted to the domain of the Web services, legible and sufficiently precise and expressive to allow the generation of executable code from it. We have used this notation for modeling the processes generated from the composed web services on orchestration mode. This modeling phase is described in detail in [7].

2.2 PHASE OF VERIFICATION AND IMPLEMENTATION

Our approach considers not only the specification of compound services but also their verification. The verification step is essential for any software development approach, it ensures the reliability and quality of the system and also helps to reduce costs since the discovery of design errors after putting into production of a system can entail significant costs. As it is better to detect errors as early as possible in the cycle of development, from the specification stage, the next step is the qualitative formal verification of our proposed model. This type of verification consist of the description of the expected behavior of the program, measured at a certain level of abstraction. The model of the system and behavioral properties described by the developer must be represented by a formal language so that they can be interpreted by formal verification tools which gives the result of verification. Our specification is described by the BPMN notation, but this language is often criticized for its lack of formality. One proposed solution is to transform the BPMN model in formal specification. Any formal specification language is susceptible to agree, but we propose the use of the process algebra LOTOS [8] which has the advantage of being supported by free formal verification tools such as CADP [9] toolbox. Due to CADP, it is possible to validate automatically the behavioral properties. In case where errors are detected, the developer is responsible for correct and refine its model to arrive at a model proven correct. The formal verification step is the object of [10] where there is more detail and description.

When the composition model is validated, the next step is the implementation of the system by generating BPEL [11] 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. More details and descriptions of this step of implementation are gives in [12].

We will provide in the next section a case study that allow to develop a composite service end-to-end using our development approach described previously.

3 Case study: E-health

Our proposed approach is based on standardized and powerful languages, templates and technologies; therefore it can solve problems of web services composition in different application domains and at all levels of complexity. We choose the health sector in view of his importance in the daily life of the citizens and for improving the quality of its services particularly to minimize the time of patient receptions and avoid blocking.

3.1 COMPOSITION SCENARIO

The following describes a typical scenario of patient journey in a hospital, in each stage of this scenario different web services can be used to inform gradually and continuously the patient's administrative and medical record.

The process is triggered by the appointment request from the patient. Once the patient is admitted, the hospital takes charge of the patient. A medical secretary reveals the patient's administrative record (if it exists, if not it creates it) and leads the patient in consultation with the doctor with whom he made an appointment and becomes the physician in charge. In order to carry out its consultation, the physician may need to access other services, such as medical record service, in order to consult and update the patient's data, status, antecedents and its history, the pharmacy service to help him to prescribe his prescription, radiology and laboratory services to request analyses and radios and receive the results. If it is necessary other services and processes can be triggered after this consultation as the hospitalization and operating block service depending on the case and the need of the patient. In all cases and at each stage the billing service is necessary to automatically appreciate the benefits and consumptions of the patient. The edition of the invoices can intervene other services like that of the insurance and the bank.

The scenario presented will be dealt throughout this document and in all stages of development.

3.2 MODELLING PHASE

3.2.1 Structure of the service composition modelling

Applying the rules and methods described in [7] and [12] to the "Online hospital" scenario, we obtain the MARDS structure shown in figure 2.

In this model of service composition, the basic components are: "Administrative Record" (AR); "Medical staff " (MS); "Appointment”; "Insurance"; "Bank"; "Medical Record” (MR); "Radiology"; "Laboratory"; "Pharmacy"; "Bed" and "Operating Block" (OB). The intermediate components are: "Home"; "Billing"; "Consultation"; "Hospitalization"; "Technical Service"(TS); "Administrative Service"(AS) and "Medical Service"(MS). The main composite component is "E-hospital".

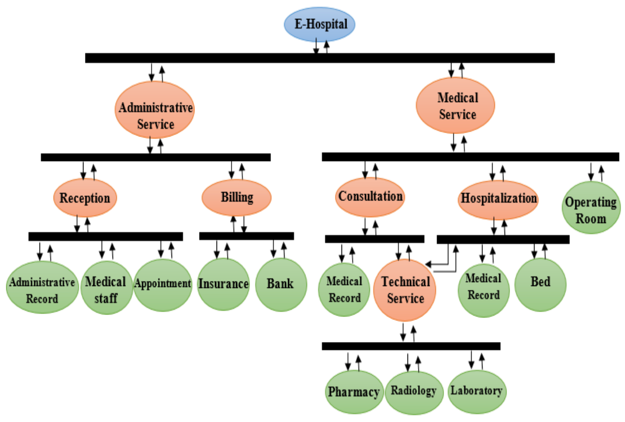

FIGURE 2 Web service composition model based on MARDS

3.2.2 Structure of the service composition modelling

Figures 3 and 4 show the business model presented in the BPMN diagram of the composition of web services.

The "A_OnlineHospital" action received by the main component (orchestrator) "E-hospital" generates two decisions {D1_ManageAdministration; D2_ManageMedicalUnit}. Each decision corresponds to a sub-action received by one of the services of the low level in the MARDS hierarchy. The first decision "D1_ManageAdministration" generates the sub-action "A_ManageAdministration". The second decision "D2_ManageMedicalUnit" generates the sub-action "A_ManageMedicalUnit".

The action "A_ManageAdministration" received by the component "AdministrativeService" generates two decisions {D1_ManagePatientInput; D2_ManagePatientOutput}. Each decision corresponds to a sub-action received by one of the services of the low level in the hierarchy. The first decision "D1_ManagePatientInput" generates the sub-action "A_ManageHome". The second decision "D2_ManagePatientOutput" generates the sub-action "A_Innvoice".

The action "A_ManageMedicalUnit" received by the "MedicalService" component generates the decision "D_ManageMedicalUnit". This decision generates three parallel sub-actions "A_ManageConsultation"; "A_ManageHospitalization" and "A_ManageOperatingBlock" received respectively by the services "Consultation"; "Hospitalization" and "OperatingBlock".

The sub-action "A_ManageHome" received by the "Home" component generates a sub-decision "D_ManageHome". On his part, this sub-decision generates three parallel sub-actions {A_IdentifyPatient; A_consultMS; A_PlanAppointment} for the components "AdministrativeRecord"; "MedicalStaff" and "Appointment". The three sub-actions correspond to the subprocess of the sub-action "A_ManageHome".

The sub-action "A_Invoice" received by the "Billing" component generates a sub-decision "D_Invoice". On his part, this sub-decision generates two parallel sub-actions {A_ConsultInsurance; A_consultBank} for the components "Insurance" and "Bank". The two sub-actions correspond to the sub-process of the "A_Invoice" sub-action.

The sub-action "A_ManageConsultation" received by the "Consultation" component generates a sub-decision "D_AchieveConsultation". On his part, this sub-decision generates two parallel sub-actions {A_ ConsultAR; A_ConsultTS} for the "MedicalRecord" and "TechnicalService" components. The two sub actions correspond to the sub-process of sub-action "A_Manage Consultation".

The "A_ManageHospitalization" sub-action received by the "Hospitalization" component generates a sub-decision "D_AchieveHospitalization". From its role this sub-decision generates three parallel sub-actions {A_AffectBed; A_ConsultMR; A_ConsultTS} for the components "Bed"; "MedicalRecord" and "TechnicalService". The three sub-actions correspond to the sub-process of sub-action "A_ManageHospitalization".

The sub-action "A_ConsultTS" received by the "TechnicalService" component generates a sub-decision "D_ConsultTS". From its role this sub-decision generates three parallel sub-actions {A_ConsultPharmacy; A_ConsultRadio; A_ConsultLabo} for "Pharmacy" components; "Radiology" and "Laboratory". The three sub-actions correspond to the sub-process of the "A_ConsultTS" sub-action.

The sub-actions {A_ManageOperatingBlock; A_ConsultPharmacy; A_ConsultRadio; A_ConsultLabo; A_AffectBed; A_ConsultMR; A_ConsultInsurance; A_consultBank; A_IdentifyPatient, A_consultMS, A_PlanAppointment} received respectively by the basic components "OperatingBlock"; "Pharmacy"; "Radiology"; "Laboratory"; "Bed"; "AdministrativeRecord"; "MedicalRecord"; "Insurance"; "Bank"; "MedicalRecord"; "MedicalStaff" and "Appointment" generate the external states {XS_OperatingBlockPlanified; XS_PharmacyResults; XS_RadioResults; XS_LaboResults; XS_BedAffected; XS_MRConsulted; XS_InsuranceResult; XS_ExpensesPaid; XS_ARcreated; XS_MSAgenda; AppointmentPlanified}.

The sub process "ManagePatientInput"; "ManagePatientOutput"; "AchieveConsultation" "AchieveHospitalization" and "ConsultTS" generate respectively the external states "HomeSet"; "InvoicePaid"; "XS_ConsAchieved"; "XS_HospAchieved" and “XS_TSConsulted".

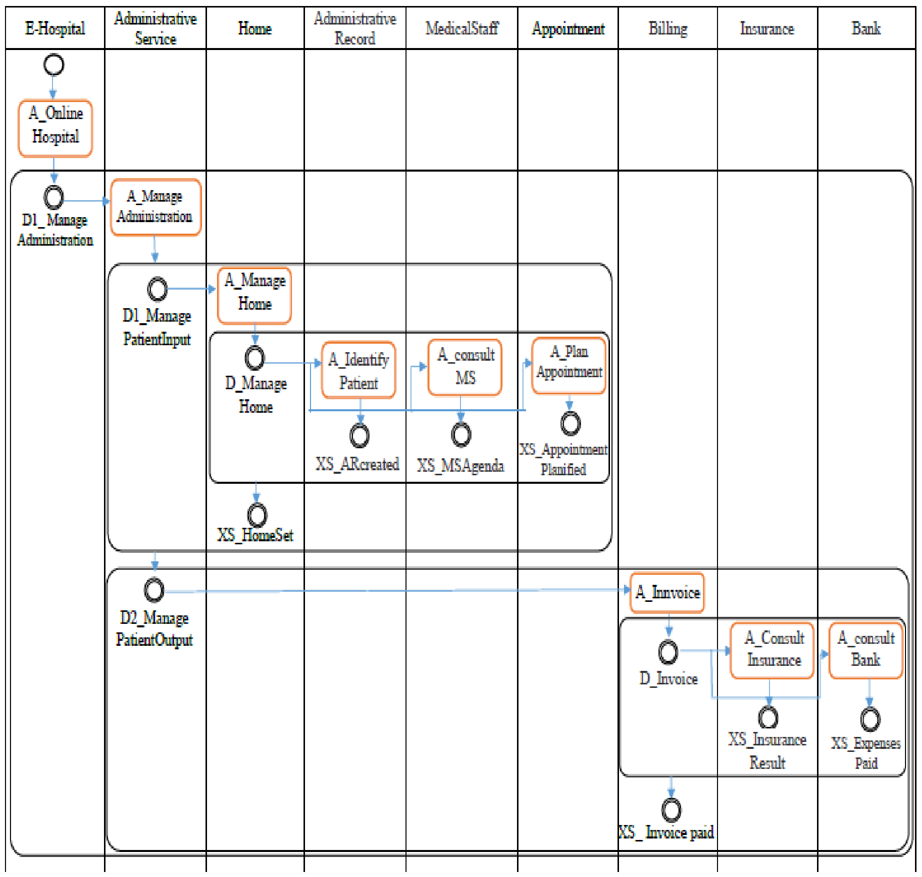

FIGURE 3 BPMN Diagram of "E-Health" Composition Scenario (Part 1)

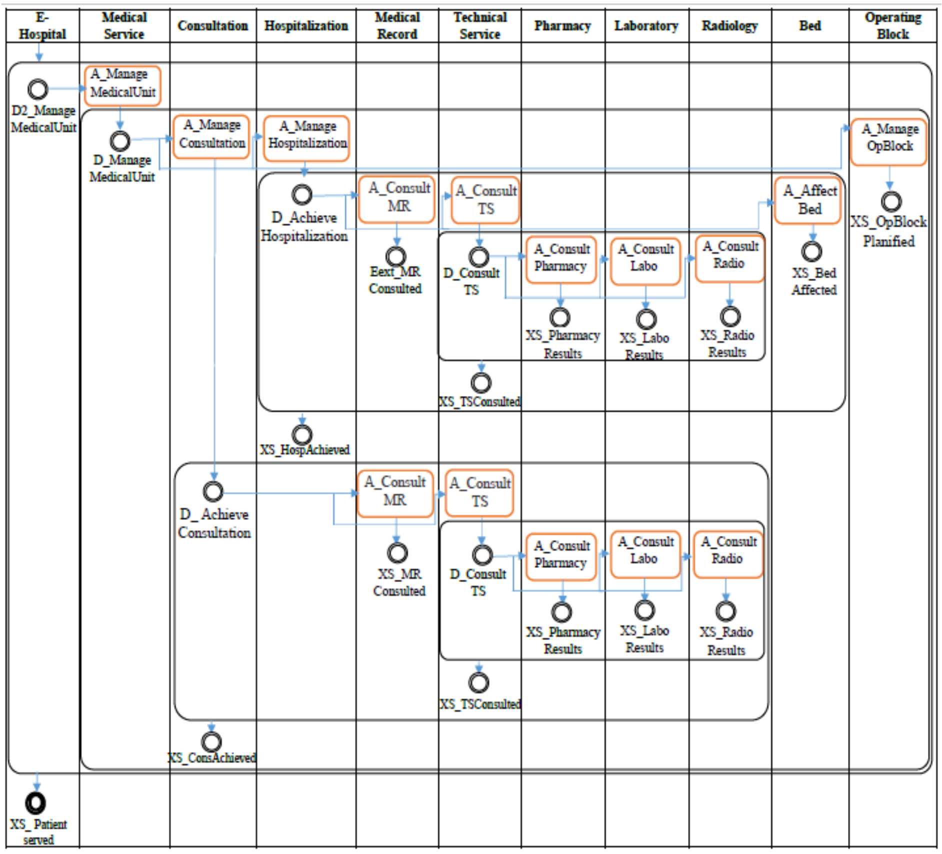

FIGURE 4 BPMN Diagram of "E-Health" Composition Scenario (Part 2)

3.3 VERIFICATION PHASE

As part of our model driven approach, service composition is expressed as a workflow or business process. This composition model can then be transformed into a formal specification described with LOTOS [14]. The goal of this transformation is to obtain a specification that can be verified formally and automatically using a tool that supports LOTOS input. CADP [15] is the most popular and successful tool for verifying models expressed with LOTOS. Indeed, the verification phase consists of two essential steps, the translation of the BPMN business model into a LOTOS specification and then the use of the CADP tool to verify the properties of the system and validate the model before embarking on the implementation phase.

3.3.1 Translation of BPMN modelling to LOTOS formal specification

To translate the BPMN notation depicted in figures 3 and 4 into LOTOS we are going to follow these steps:

- Define a process for each step of the activity (including initial and final nodes). Each process is defined by a set of behaviors.
- Assign an identifier (integer) to each of process.
- Define the gates, which are the channels of communication between processes. The peculiarity of our modeling with the SMARD model is that communication between services is done via the communication interfaces that receive and send actions and decisions, so we can consider these interfaces as processes (INTRF0, INTRF1 ... INTRFn). The actions and decisions sent and received by the services and communication interfaces are considered LOTOS gates (INPUTi, OUTPUTi) when i between 0 and n. Indeed services processes can communicate with each other through these gates, thanks to INTRF0…INTRFn processes.
- Define the operations between processes, in our example all service processes are executed concurrently using the ||| operator, which means that they are independent and they do not communicate directly with each other, but they use INTRFi process. Note however that the |[INPUTi,OUTPUTi]| operator is used to synchronize the service processes with the INTRFi process through the gates INPUTi and OUTPUTi, when i between 0 and n.
- Identify the control-flow patterns in the workflow in order to provide a definition (implementation) for each process.

The instantiation of the processes in LOTOS is provided as follow.

```
Specification Online_Hospital[INPUT,OUTPUT,
INPUT0,OUTPUT0,INPUT1,OUTPUT1,INPUT2,OUTPUT2,INPUT3,
OUTPUT3,INPUT4,OUTPUT4,INPUT5,OUTPUT5,INPUT6,OUTPU
T6,INPUT7,OUTPUT7]:noexit
behaviour
Init [INPUT, OUTPUT](0)|[INPUT,OUTPUT]| Ehospital[INPUT,
OUTPUT,INPUT0,OUTPUT0] (1)
|||
(Ehospital [INPUT, OUTPUT,INPUT0,OUTPUT0](1)
|||
AdministrativeService[INPUT0,OUTPUT0,INPUT1, OUTPUT1](2)
|||
MedicalService[INPUT0,OUTPUT0,INPUT2,OUTPUT2](3)
|||
final[INPUT0,OUTPUT0](4))|[ENV0,REC0]|
INTERF0 [INPUT0,OUTPUT0]
|||
(AdministrativeService[INPUT0,OUTPUT0,INPUT1, OUTPUT1](2)
|||
Home[INPUT1,OUTPUT1,INPUT3,OUTPUT3] (5)
|||
Billing[INPUT1,OUTPUT1,INPUT4,OUTPUT4](6))
|[INPUT1,OUTPUT1]|INTERF1 [INPUT1,OUTPUT1]
|||
(MedicalService[INPUT0,OUTPUT0,INPUT2, OUTPUT2](3)
|||
Consultation [INPUT2,OUTPUT2,INPUT5,OUTPUT5] (7)
|||
Hospitalization [INPUT2,OUTPUT2,INPUT6,OUTPUT6](8)
|||
OperatingBlock[INPUT2,OUTPUT2](9))
|[INPUT2,OUTPUT2]|
INTERF2 [INPUT2,OUTPUT2]
|||                    (Accueil
[INPUT1,OUTPUT1,INPUT3,OUTPUT3](5)
|||
AdministrativeRecord[INPUT3,OUTPUT3] (10)
|||
PersonnelMedical [ENV3, REC3](11)
|||
RendezVous [ENV3, REC3](12))            |[INPUT3,
OUTPUT3]|INTRF3 [INPUT3, OUTPUT3]                 |||
(Billing [INPUT1,OUTPUT1,INPUT4,OUTPUT4](6)
|||
Insurance [INPUT4,OUTPUT4](13)
|||
Bank [INPUT4,OUTPUT4](14))
|[INPUT4,OUTPUT4]|INTRF4 [INPUT4,OUTPUT4]
|||
(Consultation[INPUT2,OUTPUT2,INPUT5,OUTPUT5](7)
|||
MedicalRecord [INPUT5,OUTPUT5,INPUT6, OUTPUT6] (15)
|||
TechnicalService [INPUT5,OUTPUT5,INPUT6, OUTPUT6,
INPUT7,OUTPUT7](16))
|[INPUT5,OUTPUT5]|INTRF5 [INPUT5,OUTPUT5]
|||
(Hospitalization [INPUT2,OUTPUT2,INPUT6, OUTPUT6](8)
|||
MedicalRecord [INPUT5,OUTPUT5,INPUT6, OUTPUT6] (15)
|||
TechnicalService [INPUT5,OUTPUT5,INPUT6,
OUTPUT6,INPUT7,OUTPUT7](16)
|||
Bed [INPUT6, OUTPUT6](17))              |[INPUT6,
OUTPUT6,]|INTERF6[INPUT6,OUTPUT6,]
|||
(TechnicalService [INPUT5,OUTPUT5,INPUT6, OUTPUT6,INPUT7,
OUTPUT7](16)            |||
Pharmacy [INPUT7,OUTPUT7] (18)
|||
Radiology [INPUT7,OUTPUT7](19)
|||
Laboratory [INPUT7,OUTPUT7](20))
|[INPUT7, OUTPUT7]|INTRF7 [INPUT7, OUTPUT7]
where
(*Definition of process*)
endspec
```

To complete the implementation of the above specification. We go on to define the processes declared in the behaviour section. In our specification we have to define several processes, we choose some to define it in this work.

- "Init" process

The "Init" process (*Id:0*) merely starts the "Ehospital" process (*Id:1*). As a consequence, it uses the sequence pattern before exiting.

In Sequence process, an activity identified by dst_id should be executed after the completion of the activity identified by Emt_id in the workflow, we say that both activities are then executed sequentially.

```
process Init [INPUT, OUTPUT](Id:Int): exit:=
Sequence [INPUT, OUTPUT] (Id, 1)                    >> exit
where                                   process Sequence
[INPUT, OUTPUT] (Emt_Id:Int, dst_Id:Int): exit :=          ENV
!dst_Id !Emt_Id !RUN !void; exit                    endproc
endproc
```

- "Ehospital" process

The "Ehospital" process waits for a RUN message from "Init" before starting. After that, it realizes an sequence between "AdministrativeService" (*Id:2*) and "MedicalService" process (*Id:3*).

```
process Ehospital[INPUT, OUTPUT, INPUT0, OUTPUT0] (Id:Int):
exit:=                OUTPUT ! Id ! 0 of Int ! RUN ! Void;
Sequence [INPUT0, OUTPUT0] (Id,2 of Int)
>> Sequence [INPUT0, OUTPUT0] (Id,3 of Int)
>>exit                                        where
(*Definition of Sequence process*)
endproc
```

- "AdministrativeService" Process

The "AdministrativeService" process waits for a RUN message from the "Ehospital" process before starting sequentially the "Home" (*Id:5*), "Billing" (*Id:6*) processes.

```
process AdministrativeService
[INPUT0,OUTPUT0,INPUT1,OUTPUT1] (Id:Int) : exit:=
OUTPUT0 ! Id ! 1 of Int ! RUN ! Void;                    Sequence
[INPUT1,OUTPUT1] (Id,5 of Int)                        >> Sequence
[INPUT1,OUTPUT1] (Id,6 of Int)                     >>exit
where
(*Definition of Sequence process*)
endproc
```

- "Consultation" process

The "Consultation" process waits for a RUN message from the "MedicalService" process before starting concurrently the Reservation_Hairfare (*Id:4*) and "MedicalRecord" (*Id:15*) and "TechnicalService" (*Id:16*) processes, thus realizing a parallel split pattern.

In ParallelSplit process, the identifiers of the activities (*dsts_Id*) are passed in parameters to the process as a set of integers (*IntSet*). The process needs to iterate over this set and send a RUN message to each activity identified in the set. However, recursion is the only way to realize cyclical behavior in LOTOS. As a consequence, the ParallelSplit process is calling itself recursively and removing already processed *dst* from the set in order to iterate over it.

```
process Consultation [INPUT2,OUTPUT2,INPUT5, OUTPUT5]
(Id:Int) : exit:=           OUTPUT2 ! Id ! 3 of Int ! RUN ! Void;
ParallelSplit[INPUT5,OUTPUT5](Id,insert(15, insert(16, emptyset))
where                               process ParallelSplit [INPUT5,
OUTPUT5]   (Emt_Id:Int, dsts_id:IntSet) : exit :=  [empty(dsts_id)] ->
exit                                    []
[not(empty(dsts_id))] ->                                (let
dst:Int=pick(dsts_id) in                            INPUT5 !dst !
Emt_Id !RUN !void;                          BranchementMultiple [INPUT5,
OUTPUT5] (Emt_Id, remove(dst, dsts_id))            )  endproc
endproc
```

- "OperatingBlock" process

The "OperatingBlock", Like all atomic process, waits for a RUN message from other process in Upper layer before exiting.

```
Process OperatingBlock [INPUT2,OUTPUT2] (Id:Int) : noexit:=
OUTPUT2 ! Id ! 3! RUN ! Void;                              stop
endproc
```

3.3.2 Formal verification with CADP

In the second step of verification, and after the LOTOS specification is developed, we use the CADP toolkit [9] and especially the Caesar compiler in order to transform the description of the LOTOS composition into a mathematical representation in the form of a labeled transitions system (LTS) [13] on which it will be possible to verify certain behavioral properties. The developer's task is to define behavioral properties using μ-calculus [14] and verify them using the EVALUATOR compiler in CADP.

3.3.2.1 Using of Caesar compiler

Caesar is a compiler that can be used to transform a LOTOS specification into a mathematical representation. The mathematical representation used is LTS. Caesar takes the LOTOS program to check, as well as an implementation in C for the abstract types it contains (either written by hand or generated automatically by Caesar.adt). Output Caesar produces a LTS. The information contained in this LTS can be used by various tools like automation reducers, temporal or computational logic evaluators and diagnostic tools.

The LTS graph illustrated in figure 5 present the result of compilation of LOTOS specification described previously using Caesar Compiler.

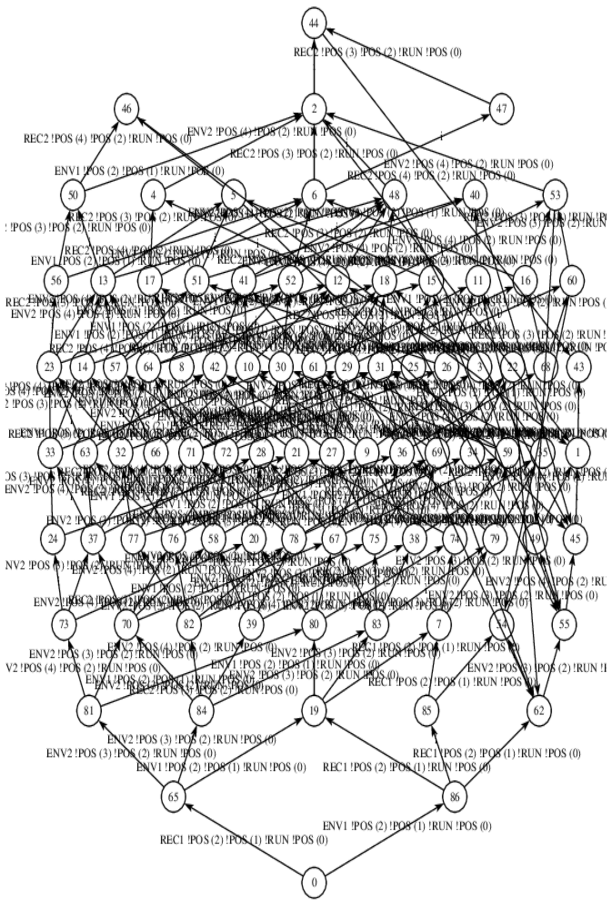

FIGURE 5 LTS generated and reduced by CADP

3.3.2.2 Using of EVALUATOR

EVALUATOR is a tool for on-the-fly verification of models integrated into CADP environment. The tool operates by taking two inputs. The first entry corresponds to the model in the form of an LTS, on which the verification is to be performed. The second input is a temporal property to be verified, expressed in the form of a formula with μ-calculus [14]. EVALUATOR will mathematically explore all possible execution branches on the generated LTS in order to prove that the property is verified (or not). The temporal property, defined by the developer and supplied to EVALUATOR, characterizes a behavior within the model. As part of our case study, we define a set of properties that would be useful to check in our example.

Property 1: We want to prove that a user can always execute the action to plan an appointment when the patient is identified and the practitioner's agenda is consulted. The property is translated to regular expression μ-calculus here as follows.

```
macro Lead (A, B) =[true_.(A)]mu X.(<true> true and [not (B)] X)
end_macro
macro IdentifierPatient() = ’ENV3!POS(10)!POS(5)!RUN.*’
end_macro
macro ConsulterAgenda() = ’ENV3!POS(11)!POS(5)!RUN.*’
end_macro
macro PlanifierRDV() = ’ENV3!POS(12)!POS(5)!RUN.*’
end_macro
Mener (IdentifierPatient and ConsulterAgenda, PlanifierRDV)
```

Property 2: We now want to show that a user can not plan an appointment without identifying the patient and consulting the practitioner's agenda. This is to verify that the inverse operation of property 1. The property is translated here as follows.

```
macro Lead (A, B) =[true_.(A)]mu X.(<true> true and [not (B)] X)
end_macro
macro IdentifierPatient() = ’ENV3!POS(10)!POS(5)!RUN.*’
end_macro
macro ConsulterAgenda() = ’ENV3!POS(11)!POS(5)!RUN.*’
end_macro
macro PlanifierRDV() = ’ENV3!POS(12)!POS(5)!RUN.*’
end_macro
Mener (not[PlanifierRDV], [ConsulterAgenda]false And
[PlanifierRDV]false)
```

The formal verification step can be repeated iteratively until a correct and refined composition model is obtained. The model can then be used as the basis for the implementation. More precisely, this is directly transformable into executable code.

3.4 IMPLEMENTATION PHASE

The implementation phase includes the generation of BPEL4WS [11] executable code from the BPMN model. Our BPMN model is based on SMARD system, therefore a description of the behaviour of a SMARD system under BPEL4WS is necessary. This description is the object of [12].

A complete BPEL description of the proposed business model should include the WSDL interfaces of the different services involved in the composition and the BPEL codes of the main process "E-hospital", the sub-processes for composite agents, and the web services performed by simple agents. As follow, we present the BPEL implementation of main process "E-hospital".

```
<!—Definition of main process of composition-->
<process name="Ehopital ">
<!-- Declaration of partnerLinks -->
<PartnerLinks>
<PartnerLink name = "Ehospital.A_OnlineHospital"
partnerLinkType = " Ehospital.A_OnlineHospital_LT"
myRole = " A_OnlineHospital_Role"
partnerRole = "A_OnlineHospital Callback_Role" />
<PartnerLink name =
"AdministrativeService.A_ManageAdministration"
partnerLinkType = "
AdministrativeService.A_ManageAdministration_LT"
myRole = " A_ ManageAdministration_Role"
partnerRole = " A_ ManageAdministrationCallabck_Role" />
<PartnerLink name = "MedicalService.A_ManageMedicalUnit"
partnerLinkType = "MedicalService.A_ManageMedicalUnit_LT"
myRole = "A_ManageMedicalUnit_Role"
partnerRole = " A_ManageMedicalUnitCallback_Role" />
</PartnerLinks>
<!-- Declaration of variables -->
<variables>
<!—Input/output for Ehospital process -->
<variable name="Action_Ehospital" messageType=" StartState"/>
<variable name="ExternalState_Ehospital" messageType=" EndState"/>
<!— Input/output for MedicalService process -->
<variable name="Action_MedicalService"    messageType="
StartState"/>
<variable name=" ExternalState_MedicalService" messageType =
"EndState"/>
<!— Input/output for AdministrativeService process -->
<variable name="Action_AdministrativeService" messageType="
StartState "/>
<variable name=" ExternalState_AdministrativeService "
messageType="EndState"/>
</variables>
<!-- Definition of process main body-->
<sequence>
<receive partnerLink=" Ehospital.A_OnlineHospital"
portType=" Ehospital.A_OnlineHospital_PT"
operation="A_OnlineHospital "
Variable="Action_Ehospital"
CreateInstance="Yes" />
<!-- Decision Name = D1_ManageAdministration -->
<Flow>
<!-- Action = “A_ManageAdministration ”, Action =
AdministrativeService -->
<sequence>
<assign>
<copy>
<from expression="A_MangeAdministration"/>
<to variable="Action_AdministrativeService" part="Message"/>
</copy>
</assign>
<invoke partnerLink=" AdministratifService.A_ManageAdministration"
portType=" AdministratifService.A_ManageAdministration_PT"
operation=" A_ManageAdministration"
inputVariable="Action_AdministrativeService" />
<receive partnerLink=
"AdministrativeService.A_ManageAdministration"
portType= "AdministrativeService.A_GererAdministrationCallBack"
operation="A_ManageAdministrationCallBack"
Variable="EtatExterne_AdministrativeService" />
</sequence>
</Flow>
<!-- Decision Name = D2_ManageMedicalUnit-->
<Flow>
<!-- Action = “A_ManageMedicalUnit”, Action = MedicalService --
>
<sequence>
<assign>
<copy>
<from expression="A_GererUniteSoin"/>
<to variable="Action_ServiceMedical" part="Message"/>
</copy>
</assign>
```

```
<invoke partnerLink="MedicalService.A_ ManageMedicalUnit"
portType=" MedicalService.A_ManageMedicalUnit_PT"
operation=" A_ManageMedicalUnit"
inputVariable="Action_MedicalService" />
<receive partnerLink=" MedicalService.A_ManageMedicalUnit"
portType=" MedicalService.A_ManageMedicalUnit CallBackPT"
operation=" A_ManageMedicalUnitCallBack"
Variable="EtatExterne_MedicalService" />
</sequence>
</Flow>
<invoke partnerLink=" Ehospital.A_OnlineHospital"
portType=" Ehospital.A_OnlineHospitalCallBack_PT"
operation=" A_OnlineHospitalCallBack"
outputVariable=" EtatExterne_Ehospital" />
</sequence>
</process>
```

5 Conclusions

In this work, an approach for the specification, formal verification and implementation of composite Web services is proposed. It is a model-driven approach faithful to the MDA principles.

Our approach of composition of web services is based on standardized and powerful languages, templates and technologies (MARDS model, BPMN Notation, BPEL language, LOTOS language, CADP tool), therefore it can solve problems of the web services composition in different application domains and at all levels of complexity. As part of a case study, we chose the health sector and we have detailed each step of our proposed approach, to better explain, illustrate and help to understand this approach.

For the prospects, we are currently working on the automatic transformation of BPMN models into LOTOS models and BPEL code. This work is important to facilitate the task of the developer and to make the steps of formal verification and implementation simple, rapid and completely automatic.

References

[1] 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

[2] Lécué F, Léger A, Delteil A 2008 DL Reasoning and AI Planning for Web Service Composition *Web Intelligence* 445–53

[3] Ying L 2010 A Method of Automatic Web Services Composition Based on Directed Graph, cmc *International Conference on Communications and Mobile Computing* **1** 527-31

[4] Berrada M, Bounabat B, Harti M 2007 Modeling and simulation of Multi-Agent reactif decisionnal systems using business process management concepts *International Review on Computers and Software (IRECOS)* **2**(2) 159-69

[5] BPMI.org. 2006 *Business Process Modeling Notation Specification. OMG Final Adopted Specification* http://www.bpmn.org/

[6] Aaroud A, Labhalla S E, Bounabat B 2005 Modelling the handover function of global system for mobile communication *The International Journal of Modelling and Simulation, ACTA Press* **25**(2)

[7] Adadi N, Berreda M, Chenouni D, Bounabat B 2014 Multi-Agent Architecture for Business Modeling of Web Services Composition based on WS2JADE Framework *International Review on Computers and Software (IRECOS)*

[8] Bolognesi T, Brinksma E 1989 *Introduction to the iso specification language lotos. The Formal Description Technique LOTOS* 23–73

[9] Fernandez J C, Garavel H, Kerbrat A 1996 Cadp-a protocol validation and verification toolbox *Proc. of International Conference on Computer Aided Verification*

[10] [7] Adadi N, Berreda M, Chenouni D 2016 Formal Specification of Web Services Composition using LOTOS *International Journal of Computer Technology and Applications (IJCTA)*

[11] OASIS Standard. Web services business process execution language version 2.0, April 2007

[12] Adadi N, Berreda M, Chenouni D, Bounabat B Modeling and Simulation of Web Services Composition Based on MARDS Model *10th International Conference on Intelligent Systems Theories and Applications SITA'15*

[13] Katoen Joost-Pieter 2005 Labelled transition systems *Model-Based Testing of Reactive Systems* **3472** 615–6 2 citations aux pages 74 et 98

[14] Emerson E A, Lei C L 1986 Efficient model checking in fragments of the propositional mu-calculus *Proceedings of the 1st LICS* 267–78 IEEE Computer Society Press 2 citations aux pages 99 et 132

AUTHORS

	Nouha Adadi, 06/06/1989, Fez Computer engineering degree from the National School of Applied Sciences (ENSA), Sidi Mohamed Ben Abdellah University, Fes, Morocco, in 2012. Major Fields: development, databases, network, security. Research interests: multi-agent system, web services, autonomic computing.
	Mohammed Berrada Received the Ph.D. degree in Computer Sciences from the faculty of Sciences, Sidi Mohammed ben Abdellah University (USMBA), Fez, Morocco, in 2008. He is currently a member of LSIS Laboratory and an Assistant Professor of Computer Sciences at ENSAF (National School of Applied Sciences of Fez), USMBA, Fez, Morocco. His current research interests include Multi-Agent systems, Enterprise Architecture, Modeling, Web services, Autonomic computing.
	Driss Chenouni Received the Ph.D. degree in physics from the University of Montpellier II, France, in 1989, and the State Doctor's degree from the University of Fes, Morocco, in 1996. He is currently a member of Laboratory of Information Science and Systems LSIS, at (E.N.S.A), and a Director of the Ecole Normale Supérieure at Sidi Mohammed Ben Abdellah University (USMBA), Fez, Morocco. His current research interests include Multi-Agent systems, Enterprise Architecture, Modeling, Web services, Autonomic computing.
	Bouchaïb Bounabat PhD in Computer Sciences. Professor in ENSIAS, (National Higher School for Computer Science and System analysis), Rabat, Morocco. Responsible of "Computer Engineering" Formation and Research Unit in ENSIAS, Regional Editor of Journal of Computing and Applications, International Expert in ICT Strategies and E-Government to several international organizations, Member of the board of Internet Society - Moroccan Chapter.