**Adapting Derived Distributed System Components**

**to Evolving Requirements**

**Mohammed Erradi, Faïçal Yannick P. Congo, …**

Networking & Distributed Systems Research Group, TIES, SIME Lab, ENSIAS, Mohamed V-Souissi University, Rabat, Morocco  
Email: [erradi@ensias.ma](mailto:erradi@ensias.ma), congo\_bf@yahoo.fr

**Abstract:** Among the challenging problems in distributed software systems engineering is to accommodate the environment changes to a designed system. Usually the changes are introduced at the high level (requirements specification level) before being propagated to the existing system components. This paper is concerned with the dynamic change of distributed systems specifications where the system activities are seen as collaborations among several components. This work is based on Bochmann’s work where an algorithm was provided to derive a distributed system components from a global requirement specification. This paper provide a Meta-Level Architecture, which consists in a set of meta-components, and an adaptation algorithm to allow for dynamic changes at the specification level and their propagation to the system component level. A set of rules is also defined to ensure a structural and behavioral conformance of the systems derived after the changes.

1. **Introduction**

There is a consensus that the construction and maintenance of large distributed software systems would greatly benefit from the existence of explicitly stated architectural principles. Such principles should provide techniques and approaches to deal with adapting the software to its environment evolutionary changes, and to govern its structure and its dynamic behavior according to such changes. Among the challenges is to find rigorous methods to support the dynamics of distributed software systems engineering which is always open to change.

The transformation algorithm presented in this paper extends the one developed by Bochmann’s work in [1]. Originally, the algorithm presented in [1] was dedicated to deriving component designs from global service and workflow specifications. This paper is mainly concerned with the dynamic change of global requirements specification of a distributed system where the system activities are seen as collaborations among several components [2,3].

\*\*\* text from [1]

Various notations have been proposed for defining the global requirement models for a distributed application system. We mention in particular UML Activity Diagrams, Use Case Maps (UCM), the Process Definition Language (XPDL) of the Workflow Management Coalition, the Business Process Execution Language (BPEL), and the Web Services Choreography Description Language (WS-CDL) developed by W3C. These different notations contain many common concepts, but also show important differences. They all have in common that the overall workflow behavior can be decomposed into several sub-activities, and further into sub-sub-activities. Most of these notations assume that the basic (primitive) activities in this behavior decomposition are activities that are allocated to a single system component within the architectural design of the system. However, for many of distributed applications, the basic building blocks of the behavior are activities that are actually collaborations between several system components, for instance a service operation between a client and a server.

Therefore [2] proposed an approach using the UML Collaborations as the basic building blocks for constructing global requirement models. Such approach was to use the sequencing operations of UML Activity Diagrams and use Collaborations as the basic activities; the temporal order among these collaborations is then defined by the flow relations of the Activity Diagrams. Before the transformation into a design model, it is important to define the architectural design and to identify the different system components that are involved in providing the different functions of the system. For each of the primitive collaboration activities identified in the global requirements model, one has to determine which system component will implement each of the collaboration roles involved. This goes hand in hand with the allocation of system resources and is very important for obtaining the desired system performance characteristics. This question of what is the best architectural design, resource allocation and allocation of collaboration roles to different system components, in short “architectural choices”, is not further developed in this paper. Instead, we concentrate here on the subsequent question: What should be the dynamic behavior of the global system components, after changing the global requirements, and how to preserve the consistency of both the structure and the behavior either at the requirement level and at the system components level.

We note that the same kind of question has been addressed by many papers during the last 10 years in a context where the global requirements are defined in terms of Message Sequence Charts (MSCs) or UML Sequence Diagrams. In this context one usually wants to describe the behavior of each system component in the form of a state machine.

\*\*\*end of text [1]

As originally defined in [1], we use a number of temporal ordering operators, similar to those found in Activity Diagrams, XPDL and BPEL, to build the global requirements model for a system. In this paper, we show how such an abstract model, together with the allocation of collaboration roles to the system components identified by the architectural design, combined with dynamic change operators, can be automatically transformed into a set of component behavior models. A set of structural and behavioral rules are also defined to provide a system that is correct by construction, that is, they will give rise to a new global system behavior, that satisfies the global requirements model, after the changes having been made.

The paper is structured as follows. After a problem statement in Section 2, Section 3 presents the static architecture where a distributed system components are derived from of the global requirements. It presents also the dynamic architecture showing the meta-component introduced to perform the dynamic change to system components after changing its requirements specification. Section 4 describes also the main concept and operators, for describing the temporal ordering of activities in the global requirements of a distributed system, as used in Bochmann’s approach.

Section 5 is considered as the main section of this paper. In Section 5.1 we present the needed initializations and the structural and behavioral requirements for a dynamic change. In Section 5.2 we describe the dynamic adaptation algorithm, and then we present the structural and behavioral conformance constraints in Section 5.3 and Section 5.4 respectively. Then some examples are discussed in Section 6, before providing our conclusions in Section 7.

1. **Problem Statement (May be : Requirements..)**

There is a consensus that the construction and maintenance of large distributed software systems would greatly benefit from the existence of explicitly stated architectural principles. Such principles should provide techniques and approaches to deal with adapting the software to its environment evolutionary changes, and to govern its structure and its dynamic behavior according to such changes. The problem of how the dynamic software architectures modify their architecture and enact the modifications during the system’s execution has been widely addressed [4,5,6]. However, one of the key issues in distributed systems dynamic evolution is how the system architecture evolve to meet changing requirements?

Usually the changes are introduced at the high level (requirements specification level) before being propagated to the existing system components at the design level and then to their running instances. This work is concerned with the dynamic change of distributed systems specifications and the corresponding evolution of the system architecture.

We consider here the case where the distributed system activities are seen as collaborations among several components.

\*\*\*\*\*\*\*\*\*

1. **Architecture**
   1. Static Derivation Process (text from Bochmann[1])

In Bochmann’s paper[1], his work was concerned with the transformation from a global requirements model, which describes the functional behavior of a distributed system in an abstract manner, to a distributed system design where the different system components are identified and their behavior must be determined such that their interactions give rise to a behavior satisfying the global requirements model. This transformation is called *design synthesis* as shown in Fig.3. 1. At the design level, the behavior of the different system components are often modeled using communicating state machines or modeling languages such as SDL or UML State Diagrams. The translation from these models into implementation code can be largely automated.



**Fig. 3.1.** A Derivation model (from [1])

\*\*\*\*\*\*\*\*\*\*\*\*\*

* 1. Dynamic Derivation Process

The key components of our dynamic derivation process architecture are: the Global System Component, the Meta Structural Component, and the Meta Behavioral Component (as shown in Fig. 3.2).

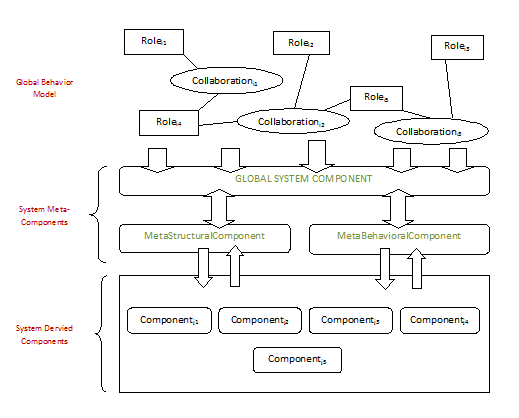


Fig. 3.2. Gobal System architecture for Dynamic Evolution

1. **Bochmann’s Approach Overview**

Our approach starts basically from the work done by G.v. Bochmann [1] which provides a formal approach and an algorithm for deriving a distributed system components from a system global behavior specification. A global specification consists in a behavior expression using the operators shown in Table 4.1. A set of calculating rules is show in Table 4.2, and the definition of the translation function Tc, the hart of the derivation algorithm, is shown in Table 4.3.

* 1. **Table1. Operators used in behavior expressions**

|  |  |  |
| --- | --- | --- |
| **Construct** | **Notation** | **Explanation of the semantics** |
| primitive  activity | <action>(r) | Execution of a local action with name <action>  performed by role r |
| invocation  of sub-col. | <subcol>(R) | Execution of a collaboration with name <subcol>  involving the set R of participating roles |
| strong  sequence | C1 ;s C2 | C2 is executed after C1 in **strong** sequence, that is, all actions of C1 are completed before C2 can start |
| weak  sequence | C1 ;w C2 | C2 is executed after C1 in **weak** sequence, that is, only local order is enforced by each participating role |
| choice | C1 [] C2 | Either C1 or C2 is executed; this may be a local choice (that is, the choice is performed by a single role / component) or competing initiatives from several roles; for a more detailed discussion, see [2]) |
| strong  while loop | C1 \* s C2 | C1 is executed zero, one or more times and then C2 will be executed; more precisely, the behavior starts with a choice between C1 and C2 ; if C1 is executed, there is **strong** sequencing between the end of C1 and the choice of executing C1 again or terminating the loop with C2 ; we assume that the choice is local (performed by a single role). |
| weak while  loop | C1 \* w C2 | As above, except that **weak** sequencing is used between  the end of C1 and the choice of executing C1 again or terminating the loop with C2 |
| concurrency | C1 || C2 | C1 and C2 are executed concurrently |
| interruption | C1 |> C2  else C3 | C1 is executed, but may be interrupted by C2 which represents a choice with priority; C2 is enabled as soon as C1 starts. If C2 does not occur (or occurs when C1 is already terminated) then C3 will occur after C1 (this is the other choice alternative). |

**Table4.2. Rules for calculating the starting, terminating and participating roles**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Starting roles (SR)** | **Terminating roles**  **(TR)** | **Participating roles**  **(PR)** |
| <action>(r) | {r} | {r} | {r} |
| <subcol>(R) | SR(<name>) | TR(<name>) | PR(<name>) = R |
| C1 ;s C2 | SR(C1) | TR(C2) | PR(C1) U PR(C2) |
| C1 ;w C2 | SR(C1) U  (SR(C2) - PR(C1)) | TR(C2) U  (TR(C1) - PR(C2)) | PR(C1) U PR(C2) |
| C1 [] C2 | SR(C1) U SR(C2) | TR(C1) U TR(C2) | PR(C1) U PR(C2) |
| C1 \* s C2 | SR(C1) = SR(C2)= {r} | TR(C2); SR(C1) if C2= ε | PR(C1) U PR(C2) |
| C1 \* w C2 | *as above* | TR(C2) U  (TR(C1) - PR(C2)) | PR(C1) U PR(C2) |
| C1 || C2 | SR(C1) U SR(C2) | TR(C1) U TR(C2) | PR(C1) U PR(C2) |
| C1 |> C2  else C3 | SR(C1) | TR(C2) U TR(C3) | PR(C1) U PR(C2) U PR(C3) |

**Table 4.3. Definition of the translation function Tc for component c**

|  |  |
| --- | --- |
| C = C1 [] C2 | Tc(C) = DOcimc(C1, C2) [] DOcimc(C2, C1)  where DOcimc(C1, C2) = if *c in Alloc(PR(C1)) then*  “( Tc (C1)” *if c is responsible for cim then*  “|| send cim(y) to all c’ in (Alloc(PR(C2)) - Alloc(PR(C1))) )” ;  else *if c in (Alloc(PR(C2) - Alloc(PR(C1))) then* “receive cim(y)“  Note: The function DOcimc(C1, C2) generates code for performing  C1, and looks after the transfer of choice indication messages from  some component participating in C1 to those components not  participating in C1, but in C2. |
| C = C1 \* s C2 | We assume Alloc(SR(C1)) = {r}, and Alloc(SR(C2)) = {r} or C2 = ε. Tc(C) =“(“ Tc(C1 ) “;“ SFM(C1 , C1) “;“ RFM(C1 , C1) “)\* ; ( “ Tc(C2) *if c=r then* “|| send cim(y) to all c’ in PR“ *if c in PR then* “|| receive cim(y) from r“ “)” *where* PR = Alloc(PR(C1)) - Alloc(PR(C2)) – {r} |
| C = C1 \* w C2 | As above, except that the SFM and RFM constructs are absent |
| C = C1 || C2 | Tc (C) = Tc (C1) || Tc (C2) |
| C = C1 |> C2  else C3 | We assume that C2 has the form “ <action>(r) ;s C2’ “.  Tc (C) = NormalBeh ||\* InterruptBeh . (see note below)  These two parts communicate within each component using the  following boolean local variables which are initially false:  Interr : an interrupt occured (but it may have occurred too late) Interrupted : the normal behavior has been interrupted  In addition, a local variable I-Enabled is used by the InterruptBeh part. The action “wait(x)”waits until the expression x becomes true.  **NormalBeh** =  *if c in Alloc(PR(C1)) then* “( Tc (C1) |> (wait(Interr);  Interrupted := true; ) else ε );”  *if c in Alloc(TR(C1)) then* “send fim(x, Interrupted) to all c’ in SR”  *if c in SR then* “(for all c’ in (Alloc(TR(C1))–{c}) do  (receive fim(x, i) from c’; if i then Interrupted := true;);  if not Interrupted then DOcimc (C3, C’2); )  ||\* (wait(Interrupted); DOcimc (C’2, C3) ) ) “  *else* “ (DOcimc (C’2, C3) [] DOcimc (C3, C’2) ); “  *where* SR = (Alloc(SR(C’2)) U Alloc(SR(C3))) –{c}  **InterruptBeh** = *if c = r then (*  *if c in (Alloc(SR(C1)) then* “I-Enabled := true; “ *else* “for all c’ in  (Alloc(SR(C1))–{c}) do (receive iem(z); I-Enabled := true)  || ( wait(I-Enabled); <action> (\* this may never happen \*) ;  Interr := true; send im(z) to all c’ in (Alloc(PR(C1)) - r) ; ) “  *else (\* c not equal r \*) (*  *if c in Alloc(SR(C1)) then “*send iem(z) to r; “  *if c in Alloc(PR(C1)) then*  ”(receive im(z) from r (\*may not happen \*); Interr := true; )” |

1. **Dynamic Adaptation Approach**

To describe the dynamic adaptation algorithm, we begin by defining the initialization step which consists in defining the needed variables and their initial values. We start by presenting the needed initializations and the structural and behavioral requirements for a dynamic change. Then we describe the dynamic adaptation algorithm. Afterwards we present both the structural and behavioral conformance rules which need to hold after the global requirement changes and their propagation to the global system components.

* 1. **Initialisation Phase**

To perform dynamic changes to an existing system, we assume that we have a global requirements specification and its corresponding derived system components. This is considered as the global system initial state before making the changes. To perform such changes we need to consider such initial state as the input of the dynamic adaptation algorithm. Therefore, such input considered as requirements constraints, will be extracted and represented in the form of matrixes. Such matrices could be derived from the global requirement specification written using UML2.0 collaborations. We have two categories of matrices: the first category, presented in Section 5.1.1, concerns the structural aspects of the initial global requirement specification, and the second category, presented in Section 5.1.2, concerns the behavioral aspects of such specification.

* + 1. **Structural Requirements**

The first category of matrices is consists in the Collaboration-Role matrix, the Component-Role matrix, the Is-Responsible-for matrix, and the Stored-Global-Behavior matrix.

***Collaboration-Role Table***

This matrix holds the roles and collaborations relationships as described in the global requirement specification. As required by the derivation algorithm, for each collaboration we need to know the type of its associated roles. A role could be a *starting*, a *participating* or a *terminating* role within a given collaboration. As shown in Table 5.1, Role1 is a starting role and participating role for collaboration1 while Rolen is a participating role in Collaboration1 and participating and terminating role in Collaborationn. We can observe that some configuration could not be possible such as (1,0,0), (0,1,0), (1,1,0). This is due to the fact that a starting role or a terminating role of a collaboration is also a participating role. However, a role could be a participating role within a collaboration without being neither a strating role nor a terminating role.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Role1 | | | … | Rolen | | |
| Starting Role | Terminating Role | Participating Role | Starting Role | Terminating Role | Participating Role |
| Collaboration1 | 1 | 0 | 1 | … | 0 | 0 | 1 |
| … | … | | | … | … | | |
| Collaborationn | 1 | 1 | 1 | … | 0 | 1 | 1 |

Table 5.1 : Collaboration\_Role matrix

***Component-Role matrix***

This matrix holds the relationships between a component and its associated roles as shown in Table 5.2. The last column of this matrix shows if the component has been derived or not. This information will be useful when performing a dynamic change. Therefore the state column indicate if a component exists already or need to be created as a new derived component. The first row indicates that existing (state=1) component1 is involved in Role1. The last row indicates that (the non derived yet: state=0) component is involved in Rolen.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Role1 | … | Rolen | State |
| Component1 | 1 | … | 0 | 1 |
| … | … | … | … | … |
| Componentn | 0 | … | 1 | 0 |

Table 5.2 : Component\_Role matrix

***Is\_Responsible\_ For matrix***

This matrix plays an important role in this model and indicates those components responsible for sending a message, during the derivation process (see Table 4.3) related to a choice construct. This concern the coordination message *Choice indication message* used for propagating the choice to a component that does not participate in the selected alternative. This message is abbreviated to cim(y) where y indicates to which choice construct the message refers. As shown in Table 5.3, the collaborations (Collaboration1, Collaboration1), from row1 and column , are not involved in a choice construct; this is designated by the “epsilon” on the table. However Collaboration1and Collaborationn are involved in a choice, where the derived Compi and Compj paly certain role in these collaborations respectively. If the first action within Collaboration1 starts, then Compi is responsible for signaling, to the components having a role in Collaborationn and not in Collaboration1, that Collaboration1 will be executed. Otherwise, if the first action within Collaborationn starts, then Compj is responsible for signaling, to the components having a role in Collaboration1 and not in Collaborationn, that Collaboration1will be executed.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Collaboration1 | | … | Collaborationn | |
| Composant1 | Composant2 | Composant1 | Composant2 |
| Collaboration1 | epsilon | epsilon | … | Compi | Compj |
| … | … | | … | … | |
| Collaborationn | Compj | Compi |  | epsilon | epsilon |

Table 5.3 : Is\_Responsible\_For matrix

***Stored Global Behaviors***

The Stored\_Global\_Behavior table is introduced to keep track of the dynamic evolution of the global requirement specification and to keep track also of the state of the derived system components during its evolution. This table, shown in Table 5.4, contains a set of the global requirement specifications for which the system components have been derived (state=1) and the new global requirement specification for which the components are under the derivation process (state=0). The first column of this table indicates the unique Id of each requirements specification, the second column indicates the behavior expression of the global requirements specification, and the third column indicates other specific information such as designerId, date, hour, etc. and the state of the derived system components. Table 5.4 shows that the oveall system has evolved from Ps1 to Psn where Psn is the requirements specification leading to a stable derived system components (state=1). Psn+1 is the last new requirements specification to be processed through the derivation algorithm which decides if the system components could be correctly derived or not.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Expression | Informations | |
|  | State |
| Ps1 | Col1 ;sCol2 |  | 1 |
| … | … | … | |
| Psn | ((Col1||Col3) ;s(Col4\*wCol2))|>Col5elseCol6 |  | 1 |
| Psn+1 | (Col1||Col3) ;s(Col4\*wCol2) |  | 0 |

Table 5.4 : Sored\_Global\_Behavior matrix

* + 1. **Behavioral Requirements**

To allow the modified system to move to a consistent state after the changes have been made, we need to preserve both structural and behavioral consistencies. The previous Section discussed the structural requirements. However a consistent structural change may lead to an inconsistent system. For example the addition of a collaboration or the addition of a role to a collaboration may lead to an undesirable situations resulting from such a change to the global requirements specification. We distinguish two categories of behavioral inconsistencies: the first one called “Coordinating Message” and the second one called “domain related”. The “Coordinating Message” inconsistency is due to the fact that a component is waiting for a coordination message that never arrives, or sending a coordination message which will never be received. The “domain related” inconsistency may occur after the addition of a role or a collaboration where an action may need the presence of another action (synchronization), or an action may enter in a conflict with another one. This may happen when we have a global requirements specification where the application domain semantics is not expressed correctly. In fact to avoid these conflicting actions situations we will introduce two matrices: the *Conflictual* matrix and the *Synchronous* matrix.

***Conflictual Matrix***

This matrix indicates for all actions pair if such actions may occur simultaneously or not. The absence of an eventual conflict between two actions is designated by 0, and the presence of a conflictual situation is designated by 1. In the presence of a conflictual situation, the conflictual actions should not be involved in a parallel composition. As shown in Table 5.5, Action1 and Actionn are conflictual actions. This mean that providing a global requirement specification Psi, where these two actions are involved within two collaborations composed by a parallel operator, leads to a “domain related” inconsistency. We assume that the designer has a good knowledge of the application domain in order to define the appropriate conflict constraints to be stored in the *Conflictual* Matrix.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Action1 | … | Actionn |
| Action1 | 0 | … | 1 |
| … | … | … | … |
| Actionn | 1 | … | 0 |

Table 5.5: Conflictual matrix

***Synchrony Matrix***

This matrix indicates which actions need to be synchronized with which other actions. If two actions synchronized have to be synchronized, then they should belong to two different collaborations composed using the parallel operator. Imposing such synchrony between actions is considered as a constraint of the application domain of the specified system. Therefore we could have a synchronous or an asynchronous behavior between the collaboration actions. This matrix is then non symmetric. Two actions in two different parallel sub-collaborations may be invoked synchronously or in an asynchronous manner. In the synchronous case, an action a1 within a sub-collaboration needs to wait for an action a2 within another sub-collaboration and vice-versa. In the asynchronous case, an action a1 within a sub-collaboration may need the execution of another action a2 within another sub-collaboration, but a2 don’t need a1 to be performed. The Table 5.5 shows the synchrony relationship between different actions.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Action1 | … | Actionn |
| Action1 | 1 | … | 1 |
| … | … | … | … |
| Actionn | 0 | … | 1 |

Table 5.5: Synchrony matrix

* 1. **The Dynamic Adaption Algorithm**

Our dynamic adaptation algorithm (Algorithm 1.) starts with an existing globlal requirement specification of a given system and its derived components. The derivation of the mentioned components is assumed to be performed using Bochmann’s derivation algorithm. Our Dynamic Adaptation Algorithm was designed to allow a dynamic change of the global requirements specification while ensuring the change propagation to the underlying derived system components. It is obvious that after performing the changes we need to obtain consistent system components. Therefore we need to ensure a structural conformance and a behavioral conformance between the initial system and the target system obtained after performing such changes.

The algorithm takes as input the existing global requirements specification Ps for which a system (derived components) s has been derived. The algorithm uses also the modified specification Ps’ for which an adapted system s’ will be dynamically derived. Note that s’ is obtained from s after the changes introduced in the new global requirement specification Ps’. These changes may lead to the addition/deletion of new components and/or their associated coordinating messages. In addition the algorithm uses also: (a) the structural and behavioral matrixes described previously in Section 5.1, and (b) the meta actions: MCreate(x), MDelete(x) and MUpdate(x), where x is the couple (c,Tc(Ps’)) (c is the component and Tc(Ps’) is its associated behavior.

The algorithm starts by detecting if a change request is made by comparing the behavior expression of the specification **Ps** with the behavior expression of **Ps’**. The change request could be detected also if the Collaboration\_Role table or the Component\_Role table has been changed. Then the algorithm proceeds to ensure that the requested changes will not violate the structural and behavioral constraints. The structural constraints are defined to be able to ensure a structural conformance of the new requirements specification **Ps’** with the collaboration based model (see Table 4.1). This consists in checking the syntactic conformance of **Ps’**, and the conformance of the specified collaborations. For instance, a collaboration should not exist without having at least one participating role.

Then the algorithm checks for the behavioral conformance of the new global requirements specification **Ps’** with respect to the behavioral constraints. These constraints are grouped in two categories: the first one deal with coordinating messages and the second one deal with the application domain constraints. The coordinating messages conformance sub algorithm (*CoordinationMessage*) will check if a component will not send a coordinating message (*cim, fmx, fim, im*), as shown in Table 4.3, to other components that are not supposed to receive such message. It checks also if no component is waiting for a coordinating message that no other component will send it. The *DomainRelated* sub algorithm imposes to the derived system components to stay conform to the application domain constraints as imposed by the *conflictual* matrix and the *synchrony* matrix.

After the analysis of the changes request, using the structural and behavioral conformance sub algorithms, the dynamic adaptation algorithm proceeds to the changes propagation to the system components. Therefore to propagate the changes to the existing derived system components, according to the changes made to the global requirements specification **Ps**, the algorithm proceeds as follows: for each component c in the component table the algorithm compares the behavior expression of this component in **Ps** with its behavior expression in **Ps’**. If these behavior expressions are the same, the algorithm do nothing and the component behavior remains unchanged. However if such behavior expression are not the same, this implies that a change is requested. Then if the component c have no behavior expression before the change request but it has a new behavior within the global requirements specification **Ps’**, then the component will be created including its associated behavior expression. This creation will be made by sending the meta actions *CreateM*(c,Tc(Ps’)) to invoke the *MetaUpdateStructure* meta component (shown in Fig.3.2).

In the case where the component c exists already and has a non empty behavior expression, but c has an empty behavior expression according to the new global requirements expression (i.e. Tc(Ps’)=epsilon), then c will be deleted. The deletion will be made by ending the meta action *DeleteM*(c) to the *MetaUpdateStructure* meta component. However, if the component c has a non empty behavior expression in both **Ps** and **Ps**’, then the behavior of c will be updated by sending the meta action *UpdateM*(c,Tc(Ps’)) to the *MetaUpdateBehavior* meta component.

The coordination between a new component or an existing component having a new behavior expression, and the other system components will be made according to the behavior expression of the component Tc(Ps’). This will not lead to an undesirable behavior because the behavioral conformance of such component was previously checked by the algorithm.

**Algorithm1 MétaT**: Dynamic Adaptation Algorithm

**Inputs :**

**Ps**  : the existing global requirements specification

**Ps’** : the new global requirements spcification

**Body :**

**if** **Ps**<> **Ps’** Or **Coll-Role-Table(changed)** Or **Component-Role-Table(changed)** then

**if** **StoredBehaviorsMatrix(Ps’).state** = 0 then

% Structural Conformance %

**SyntaxConformance(Ps’)**.

**CollaborationConformance(Ps’)**.

% Behavioral Conformance %

**CoordinatingMessageConformance(Ps’)**.

**DomainRelatedConformance(Ps’)**.

%End of the Change Transaction%

**Commit** := 1

% Changes Propagation %

**for** all **c** in **Component-Role-Table.Component** do

**if** **Commit** = 1 then **break for**

**if** **Tc(Ps)** = **Tc(Ps’)** then **ε**

**else**

**if Tc(Ps)** = **ε** and **Tc(Ps’)**<> **ε** then

**send CreateM(c,Tc(Ps’))** to **MetaUpdateStructure**

**else** **if Tc(Ps)**<> **ε** and **Tc(Ps’)** = **ε** then

**send** **DeleteM(c)** to **MetaUpdateStructure**

**else**

**if** **c.State()** = 0 then

**send CreateM(c,Tc(Ps’))** to **MetaUpdateStructure**

**else**

**send** **UpdateM(c,Tc(Ps’))** to **MetaUpdateBehavior**

* 1. **Structural Conformance**

C’est l’ensemble des règles de préservation de la consistance du formalisme utilisé pour représenter le comportement global du système. Elles sont décrites par des algorithmes qui sont lancés par l’algorithme principal dans la section Confirmation de la consistance transactionnelle pour confirmer ou invalider la consistance d’une nouvelle expréssion globale.

* + 1. **Syntax Conformance**

La règle de consistance syntaxique stipule que tous opérateurs dans une expression doit avoir une collaboration de part et d’autre sinon il y a violation syntaxique. Ce qui suit est l’algorithme de détection et de correction des problèmes de consistance syntaxique. (algorithm non incluse ici)

* + 1. **Collaboration Conformance**

Dans cette partie nous repérons les cas d’inconsistance en terme de collaboration. La principale est « Toute collaboration doit avoir au moins un role participatif ». L’agorithme suivant détecte et supprime les Collaborations sans roles dans Ps’ (algorithme non inclus ici).

* 1. **Behavioral Conformance**

Il s’agit ici de présenter l’ensemble des règles de préservation conportemental du système entier. En effet, pour pouvoir anticiper les problèmes de blockage comportementaux du système, il faut être capable des les détecter et tenter de proposer une solution. Ainsi, cette section exprime bien le fait d’imposer des regles de fonctionnement liées au domaine d’application. Cela permet de ce fait de dériver expressément des blockages à la demande.

* + 1. **Coordinating Messages Conformance**

Cette partie permet d’exprimer un blockage comportemental succeptible de naître du fait de la modification de certaine forme d’expréssion. Dans le travail du professeur Bochmann il s’agit des expressions entrainant la naissance de méssages de coordination dans le comportement des composants. Ceci, parcequ’avec le formalisme et la méthode de dérivation deux cas de figure sont très vite observable. Il s’agit de composant envoyant des méssages et aucuns n’en recevant ou des composants attendants des méssages de composants n’en envoyant plus. Ainsi les expréssions succeptible de générer ce type d’inconsistence sont les suivantes :

* Col1 ;s Col2
* Col1 \*s Col2
* Col1 [] Col2
* Col1 |> Col2 else Col3

L’algorithme suivant permet de détecter et de corriger les cas succeptible de générer des inconsistences. (Voir Annexe A (A.2))

Nous définissons une procédure qui détermine la forme d’une collaboration à un niveau de 1. Nous la notons Form(Col) qui peut être Col1 ;s Col2 ou Col1 \*s Col1 ou Col1 [] Col2 ou Col1 |> Col2 else Col3 ou Col1 ;w Col2 ou Col1 \*w Col1.

* + 1. **Domain Related Conformance**

Dans cette partie nous présentons le cas de blockage lié à une inconsistance entre deux ou plusieurs actions. En effet, c’est grace à ce type de règle de conflits que l’on pourra traduire les règles de fonctionnement propre au secteur d’application du système dérivé. On dira par exemple qu’en sécurité que pour pouvoir avoir accès à une donnée il faut être Auhtoriser. Ce qui signifit que l’action accessData(D1) requiert l’action Authorization pour s’achever correctement. Et à juste titre Authorization nécéssite Authentification pour aussi être menner à bien. Ainsi si le système distribué est un système de sécurisation alors ces règles là sont tout à fait exprimable et vous conviendrez avec moi que si elles ne le sont pas alors il y aura une inconsistence très forte qui fera par exemple que tout le monde pourra accéder à la donnée qu’il veut sans se soucier d’y avoir le droit ou pas. Pour ce faire nous utiliserons les matrices de synchronisation et de conflits pour détecter et essayer de résoudre ce type d’inconsistance.

Nous introduisons ici une nouvelle notion. Il s’agit de l’arbre d’activité. En fait pour chacun des roles de Ps’ il faut dériver son arbre d’actions. C'est-à-dire analyser l’expéssion en générant la succession des actions dans lesquels le role est impliquer. Nous proposons donc d’abord un algorithm de génération des arbres d’activité (Voir Annexe A (A.1))

1. **Conclusion**

- we have presented an approach concerned with the dynamic change of distributed systems specifications where the system activities are seen as collaborations among several components. We proposed an algorithm to adapt dynamically the global requirements specification changes to the existing derived system components. We provide also the way to preserve the structural and behavioral conformance of the new derived system.















































































- We are working on a reflective version of our approach in which the meta level system will made of roles and collaborations and will be written in the same language. This allows to even make dynamic changes to both the original specification language and to the dynamic adaptation components.

- implementation en cours using aspects (groovy, aspect)

- security policies

References

[1]G.v. Bochmann, Deriving component designs from global requirements Proc. Intern. Workshop on Model Based Architecting and Construction of Embedded Systems (ACES), Toulouse, Sept. 2008.

[2] H. Castejón , G.v. Bochmann, R. Bræk, Using Collaborations in the Development of Distributed Services, submitted for publication.

[3] H. Castejón, R. Bræk, G.v. Bochmann, Realizability of Collaboration-based Service Specifications, Proceedings of the 14th Asia-Pacific Soft. Eng. Conf. (APSEC'07), IEEE Computer Society Press, pp. 73-80, 2007.

[4] I. Georgiadis, J. Magee, and J. Kramer. Self-organising software architectures for distributed systems. In Proc. of the 1st Work. on Self-Healing Systems (WOSS’02), pages 33–38. ACM Press, 2002.

[5] J. S. Bradburya, J. R. Cordyay, J. Dingela, M. Wermelingerbz A Survey of Self Management in Dynamic Software Architecture Specifications. In Proc. of the International Workshop on Self-Managed Systems (WOSS'04), Newport Beach, California, USA, October/November 2004

[6] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. IEEE Trans. on Software Engineering, 26(1):70–93, 2000.

Reste à completer.

**Annexe A**

**A.1.**

**Algorithm5** : ActionsTreeGenerating Algorithme

**Input :**

**Role** : C’est le role dont on veut dériver l’arbre.

**Col** : C’est une collaboration.

**Body :**

**if Form(Col)** = ”**<action> (r)**“ then

**if Role** = **r** then

action

**else if** **Form(Col)** = “**<sub-coll> (R)**” then

**if Role** in **R** then

Sub-coll

**else if Form(Col)** = ”**Col1 ;s Col2**” Or **Form(Col)** = ”**Col1 ;w Col2**” then

**if Role** in **PR(Col1)** then

**if Role** in **PR(Col2)** then

ActionsTreeGenerating(Role,Col2)

ActionsTreeGenerating(Role,Col1)

**else**

ActionsTreeGenerating(Role,Col1)

**else**

**if Role** in **PR(Col2)** then

ActionsTreeGenerating(Role,Col2)

**else if** **Form(Col)** = ”**Col1 \*s Col2** “ Or **Form(Col)** = ”**Col1 \*w Col2**“ then

**if Role** in **PR(Col1)** then

**if Role** in **PR(Col2)** then

**Generate**

ActionsTreeGenerating(Role,Col1)ActionsTreeGenerating(Role,Col1)

ActionsTreeGenerating(Role,Col2)

**else**

**Generate**

ActionsTreeGenerating(Role,Col1) ActionsTreeGenerating(Role,Col1)

**end if**

**else**

**if Role** in **PR(Col2)** then

**Generate**

ActionsTreeGenerating(Role,Col2)

**end if**

**end if**

**else if Form(Col)** = ”**Col1 [] Col2**” then

**if Role** in **PR(Col1)** then

**if Role** in **PR(Col2)** then

**Generate**

ActionsTreeGenerating(Role,Col1) ActionsTreeGenerating(Role,Col2)

**else**

**Generate**

ActionsTreeGenerating(Role,Col1)

**end if**

**else**

**if Role** in **PR(Col2)** then

**Generate**

ActionsTreeGenerating(Role,Col2)

**end if**

**end if**

**else if Form(Col)** = ”**Col1 || Col2**” then

**if Role** in **PR(Col1)** then

**if Role** in **PR(Col2)** then

**Generate**

ActionsTreeGenerating(Role,Col1) ActionsTreeGenerating(Role,Col2)

**else**

**Generate**

ActionsTreeGenerating(Role,Col1)

**end if**

**else**

**if Role** in **PR(Col2)** then

**Generate**

ActionsTreeGenerating(Role,Col2)

**end if**

**end if**

**else if Form(Col)** = ”**Col1 |> Col2 else Col3**“ then

**if Role** in **PR(Col1)** then

**if Role** in **PR(Col2)** then

**Generate**

ActionsTreeGenerating(Role,Col2)

**else**

**if Role** in **PR(Col3)** then

**Generate**

ActionsTreeGenerating(Role,Col1)

ActionsTreeGenerating(Role,Col3)

**else**

**Generate**

ActionsTreeGenerating(Role,Col1)

**end if**

**end if**

**else**

**if Role** in **PR(Col2)** then

**Generate**

ActionsTreeGenerating(Role,Col2)

**else**

**if Role** in **PR(Col3)** then

**Generate**

ActionsTreeGenerating(Role,Col3)

**end if**

**end if**

Présentons à present l’algorithm de correction des conflits à partir des arbres de derivations et des matrices.

Nous définissons deux procédures baser sur les Arbres.

En effet **ActionsTreeGenerating(Role,Coll)** retourne un arbre que nous notons **Arbre**. Et soient :

**Abre.findParallel()** : qui retourne un ensemble de sous-ensemble d’actions en parallèle dans Arbre que nous notons **ParallelActions**.

**Arbre.findChoice()** : qui retourne un ensemble de sous-ensemble d’actions en choix dans Arbre que nous notons **ChoiceActions**.

**Arbres** : est une structure qui contient tous les arbres générer. Et **Arbres(Role)** retourne l’arbre des actions du role **Role**.

**Generate** : Procédure qui génère une portion de l’arbre

A.2.

**Algorithm6** : CoordinatingMessageConformance

**Input :**

**Ps’** : Expréssion globale à dériver

**Body :**

% Génération des arbres d’actions %

**for** all **Role** in **PR(Ps’)** do

**Arbres.add(Role,ActionsTreeGenerating(Role,Ps’))**

% Résolution de conflits intra-Role %

**for** all **Role** in **PR(Ps’)** do

**Arbre** := **Arbres(Role)**.

**for** all **action** in **Arbre.findParallel()** do

**Actions** := **ε**

% Parallel Conflict Resolution %

**CountPoint** := **0**

**if** **Conflictual-Matrix(action)** <> **ε** then

**Actions** <- **Conflictual-Matrix(action)**.

**ActionsConflictParallel := ε**

**for** all **act** in **Actions** do

**if** act in **Arbre.findParallel()\{action}** then

**CountPoint** := **CountPoint**+1.

**ActionsConflictParallel**<- **act**.

**if** **CountPoint** <> 0then

%Changer l’operateur || en ;w %

**for** all ac in **ActionsConflictParallel** do

**ChangeOperator(ac, action,’;w’)**

% Synchronous Necessity Resolution %

**CountPoint** := **0**

**if** **Synchronous-Matrix(action)** <> **ε** then

**Actions** <- **Synchronous-Matrix(action)**.

**ActionsConflictSynchron := ε**

**for** all **act** in **Actions** do

**if** **act** in **Arbre.findChoice()\{action}** then

**CountPoint** := **CountPoint**+1

**ActionsConflictSynchron**<- **act**.

**if** **CountPoint** <> 0then

%Changer l’operateur ;s ou ;w ou \*s ou \*w en || %

**for** all **ac** in **ActionsConflictSynchron** do

**ChangeOperator(ac, action,’||’).**

% Résolution de conflits inter-Role %

**for** all **Role** in **PR(Ps’)** do

**for** all **action** in **Role.getActions()** do

**for** all **r** in **PR(Ps’)\{Role}** do

**for** all **ac** in **r.getActions()** do

**if** **Conflictual-Matrix(action,ac)** then

**for** all **comp** in **Alloc(r)** do

**Include**[“**send** and **receive** **MDBA (action,ac,Alloc(r))”)**]

**Note.**

**MDBA(action,ac,Alloc(r))”)** : [MessageDeepBlockingAvoidance] qui contient soit “wait for the component Alloc(r) before doing ac because of action”.

**Include** : Permet d’ajouter au comportement du composant qu’il doit être prêt à comprendre MDBA ou bien que avant de faire action il faut d’abord ne pas avoir recu MDBA avec un « wait for compi before doing action ».

**ChangeOperator(ac, action,’||’)** : permet de changer l’operateur le plus proche entre les actions ac et action.

**Conflictual-Matrix(action)** : Procédure qui retourne les actions qui ont comme valeur 1 dans la ligne action de la matrice ConflictualActionsMatrix.

**Conflictual-Matrix(action,ac)**: Prodédure retournant vrai ou faux si deux ations ont pour valeur 1 dans la ConflictualActionsMatrix.

**Synchronous-Matrix(action)** : Procédure qui retourne les actions qui ont comme valeur 1 dans la ligne action de la matrice SynchronousActionsMatrix.

**Annexe B**

1. Study Case (Application Example) Telemed
   1. Decription

Soit un environement hospitalié constitué de plusieurs acteurs comme : le médécin, le patient , … etc.

Au premier abords nous volons que notre système se comporte de la manière suivante :

A patient is being treated over an extended period of time for an illness that requires frequent tests and consultations with a doctor at the hospital to set the right doses of medicine. Since the patient may stay at home and the hospital is a considerable distance away from the patient’s home, the patient has been equipped with the necessary testing equipment at home. The patient will call the hospital on a regular basis to have remote tests done and consult with a doctor. A consultation may proceed as follows: The patient calls the telemedicine reception desk to ask for a consultation session with one of the doctors. The receptionist will register the information needed, and then see if the doctor is available. If the doctor is available, the patient will be assigned to the doctor and the consultation can start. Otherwise, the patient is put on hold, possibly listening to music, until a doctor is available. If the patient does not want to wait any longer, he/she may hang up (and call back later). The consultation itself is defined by a separate diagram and consists of a voice communication collaboration during which certain tests are invoked using some test equipment at the patient’s location, but controlled by the doctor over distance.

Dans un premier temps nous considérons le travail du professeur Bochmann est le cas statique.

Dans un second temps et vous conviendrais que cela se passe très souvent, nous volons inclure de nouveau éléments dans notre système pour qu’il ait un autre comportement.

En effet, considérant l’état précédent du système trop basique, l’hopital a acheter un équipement de santé très sophistiqué qui permet de mésurer les constantes corporelles d’un être humain(T°C, Pression artérielle, Poul,…) mais aussi de doser les médicaments qui sont liés à lui et qu’il transmet dans le sang du malade directement.

Le nouveau scénario voulu est le suivant : L’appareil lors de son réglage est programmer pour mésurer les constantes corporelles du patient. Cet appareil va ensuite envoyer un message à une application web chez la receptionniste. Celle-ci, soit elle fait attendre l’équipement puis quand un médécin est disponible, elle envoie les relevés à celui-ci et signale à l’équipement que les données sont en cours de traitement ou bien si un médécin est disponible, elle lui envoie directement les relevés et signale à l’équipement que le relevé est en cours de traitement. Cependant si l’attente dépasse un certain temps l’équipement peut reenvoyer le méssage plustard. Un cas exceptionnel qui exprime la puissance de l’équipement est basé sur les standards de médécine des constantes corporelles. L’équipement réagit a ceci et un message d’urgence à la réceptionniste qui informe un agent de l’ambulance qui va donc évaccuer le patient (cas Température> 40°C,…). Dans le cas normal lorsque la réceptionnistre trouve un médécin et lui envoie les relevés du patient celui-ci a suffisament d’information sur la maladie, les constantes corporelles et les dosages actuels pour déterminer les dosages a prendre en compte. Il retourne alors à la réceptionniste un méssage des dosages à appliquer qui elle l’achemine à l’équipement. L’équipement change alors les dosages vers les nouvelles valeurs

Dans le cas dynamique nous montrons le passage du système de son état précédent au nouvel état précédemment décrit.

Finallement, l’équipement sofistiqué possède de grosse faïlle de sécurité et de construction. En fait on s’est apercu qu’un autre type d’actes criminels à apparu. Certains malades, mourrait sans raison apparente parceque l’équipement appliquait des dosages dangereux et n’informait pas l’hopital de l’état critique des patients. Des experts en criminologie ont mené une enquête et découvert des pirates informatiques ont pu accéder aux équipements de ces patients là et causer volontairement leurs morts. Certains de ces malades étaient des témoins clés dans des jugements de grands criminels. L’hopital poursuivit en justice par les parents des décéder et après avoir réussit à être disculpé La compagnie NovaHealth constructrice de l’équipement a débourser 12 millards de dollards pour dédomagements .L’équipement a donc été retiré du marché jusqu’à ce que les normes de sécuité les plus drastiques soient respecté. L’hopital est donc contraint de ce fait de retourner vers son ancien état avant l’installation de l’équipement. Ce changement sera exprimer aussi dans le cas dynamique.

* 1. Static Case

Referring to the example discussed in Section 2.3, let us assume that the roles P (patient), R (receptionist) and D (doctor) are to be implemented on three different components, also called P, R and D, respectively. This means that Alloc(P)=P, Alloc(R)=R and Alloc(D)=D. In the following we explain how the algorithm described in Section 3 can be used to derive the behavior of these components such that they realize the correct coordination of activities among these three components. We start out with the definition of the collaboration behavior given in Section 2.3, as follows:

<w>{P, sRt} = <wait>{Pt, sR} \*w ε

<act>{Pt, sRt, Dt} = <assign>{sRt, D} ;w <consult>{Pt, sDt}

<telemed> = <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } )

Let us first determine the behavior for the sub-activities <w> and <act> at each of the three components:

TP (<w>) = TP (<wait>) \* ; receive cim(y) from R

TP (<act>) = TP (<consult>) (\* P is not involved in <assign> \*)

TR (<w>) = TR (<wait>) \* ; send cim(y) to P

TR (<act>) = TR (<assign>) (\* R is not involved in <consult> \*)

TD (<w>) = ε

TD (<act>) = TD (<assign>) ; TD (<consult>)

For the sub-collaboration <act> for example, the rule of Table 4 indicates that TP (<act>) = TP (<assign>) ; TP (<consult>), however, TP (<assign>) is empty since the component P (and the role P) is not involved in the <assign> sub-collaboration. For the behavior of component P for the sub-collaboration <w>, we have to evaluate the rule of Table 4 for C = C1 \* w C2 where C1 = <wait>{Pt, sR} and C2 = ε . Since *“ P in Alloc(SR(C1))”* is false and *“ P in (Alloc(PR(C1)) - Alloc(PR(C2)))”* is true,we get the expression given above.

Now let us determine the behaviors of the three components for the <telemed> activity. Applying the rules of Table 4, we obtain the following behaviors for all components c = P, R or D:

Tc (<telemed>) = Tc (<registr>) ; Tc (<w> |> <h-up>; ε else <act> )

= Tc (<registr>) ; ( NormalBeh c  | |\* InterruptBeh  c )

where TD (<registr>) = ε and the behaviors NormalBeh c  and InterruptBeh c are defined as follows:

NormalBeh P  = (TP (<w>) |> ( wait(Interr); Interrupted ;= true;) else ε);

( receive cim(y) from R [] TP (<act>) )

InterruptBeh P = receive iem(z) from R; <h-up>; Interr := true; send im(z) to R

NormalBeh R  = (TR (<w>) |> ( wait(Interr); Interrupted ;= true;) else ε);

( receive fim(x, i) from P; if i then Interrupted := true; if not Interrupted

then TR (<act>) ) | | (wait(Interrupted); send cim(y) to D and P )

InterruptBeh R = send iem(z) to P; receive im(z) from P; Interr := true

NormalBeh D  = TD (<act>) [] receive cim(y) from R

InterruptBeh D = ε

Let us look at this derivation in more detail. First we note that the <telemed> collaboration is the weak sequential execution of the registration <registr>{sPt, R}followed by an interruption behavior C = C1 |> C2 else C3 , where C1 = <w>{P, sRt }, C2 = <h-up>{sPt}, and C3 = <act>{Pt, sRt, Dt}. We see that C2 consists only of the interrupt, namely <h-up>; therefore C’2 = ε, and the role performing the interrupt is r = P. The rule forr weak sequencing in Table 4 leads to the first expression for Tc (<telemed>) given above. The rule for the interruption behavior in Table 4 leads to the second expression for Tc (<telemed>) above and the different forms of NormalBeh c and InterruptBeh c for the three components listed above.

Let us look at the derivation of the expression for NormalBeh c in the case of the component c = P. We have that *“P in Alloc(PR(C1))”* is true, *“P in Alloc(TR(C1))”* is false, *“P in ( Alloc(SR(C’2)) U Alloc(SR(C3)) )”* is also false since Alloc(SR(C’2))= {}. We therefore have to determine “DOcimP (C’2, C3) [] DOcimP (C3, C’2)”, which means “DOcimP (ε, <act>{Pt, sRt, Dt}) [] DOcimP (<act>{Pt, sRt, Dt}, ε)”. The definition of the DOcim function in Table 4 for choice rule indicates for “DOcimP (ε, <act>{Pt, sRt, Dt}) “ that the “else” part applies (since the first argument is ε which includes no participating roles); this leads to the generated code “receive cim(y) from R “ since P is in Alloc(PR(<act>{Pt, sRt, Dt})) and if we select R as the component responsible to send the choice indication message to the other components when the interrupt leads to the execution of C’2 (which is empty) instead of the <act> sub-collaboration. For “DOcimP (<act>{Pt, sRt, Dt}, ε)” we obtain “TP (<act>)” since P is in Alloc(PR(<act>{Pt, sRt, Dt})). Putting all these things together leads to the expression for NormalBeh P given above.

For the expression for InterrBeh c in the case of the component c = P 2, we have the case c = r, and the condition of the next IF statement is false. Therefore we obtain the generated code defined by the “else” part of the form “ (receive iem(z) from R ; I-Enabled := true) || ( wait(I-Enabled); <h-up>; Interr := true; send im(z) to R; ) “. However, this can be simplified to “ receive iem(z) from R ; <h-up>; Interr := true; send im(z) to R; “ as given above.

By substituting the behaviors of the sub-activities <w> and <act> given above, we obtain three behavior expressions for the three system components P, R and D. These expressions include the local behaviors of the primitive collaborations <wait>, <assign> and <consult> and are independent of their particular nature; the expressions only depend of the sets of starting, terminating and participating roles given above. We note that these behaviors can also be represented by UML Activity Diagrams; for instance, Figure 6 shows the behavior for component P. We note that P is not a starting role of the <consult> sub-collaboration; therefore its behavior for this sub-collaboration will begin with the reception of some message. The choice between Tp<consult> and “receive cim(y)” at the component P will therefore depend on which message will be received by the component.

Let us now assume that the basic activities consist of the following message exchanges:

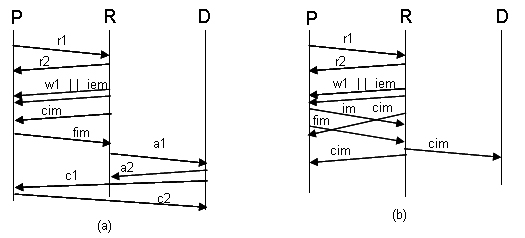
<registration> = r1 from P to R; r2 from R to P.

<wait> = w1 from R to P.

<assign> = a1 from R to D; a2 from D to R.

<consult> = c1 from D to P; c2 from P to D.

Then the above definitions of the component behaviors give rise (among others) to the execution scenarios shown in Figure 7.



**Fig. 7.** Possible execution scenarios for the telemedicine application: (a) normal behavior, (b) the user quits the waiting loop \*\*\* update: no fim message \*\*\*

* 1. Dynamic case
     1. Initial Case
        1. Initialisation

*Global Behavior Expression*

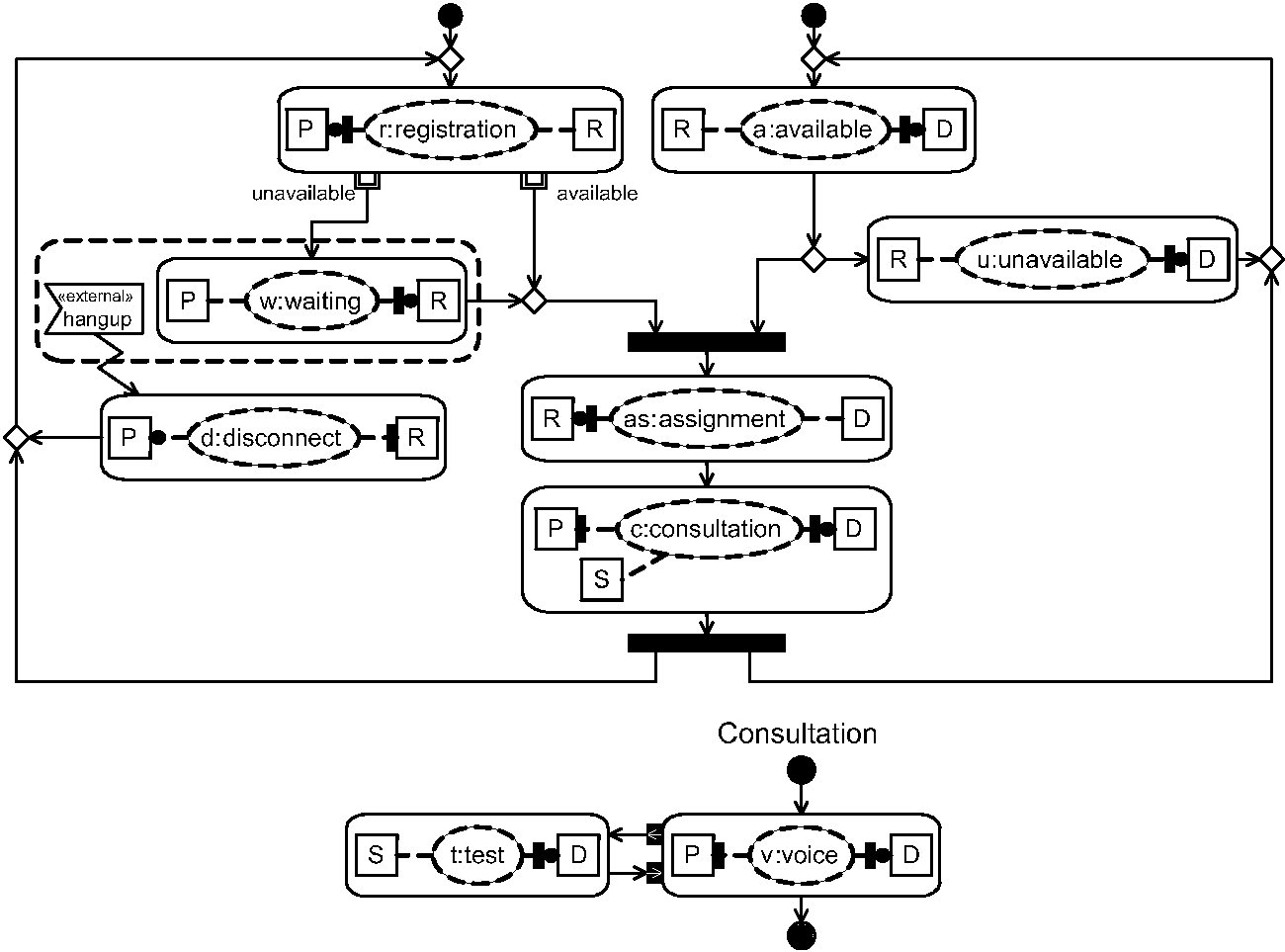
<w>{P, sRt} = <wait>{Pt, sR} \*w ε

<act>{Pt, sRt, Dt} = <assign>{sRt, D} ;w <consult>{Pt, sDt}

<telemed1> = <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } )

*UML Activity Diagram*

On présente ici le diagramme de collaboration activités correspondant au Global Behavior Expression.



**Fig. 2.** Order of collaboration activities within the telemedicine example application (taken from [1])

*Matrix Generation*

Collaboration-Role-Table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Patient | | | Doctor | | | Receptionist | | |
| SR | TR | PR | SR | TR | PR | SR | TR | PR |
| Registr | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| W | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| Act | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| h-up | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

Composant-Role-Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Patient | Doctor | Receptionist | State |
| Patient | 1 | 0 | 0 | 0 |
| Doctor | 0 | 1 | 0 | 0 |
| Receptionist | 0 | 0 | 1 | 0 |

Stored-Global-Behaviors

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Expréssion | Informations | | |
| Date | Hour | State |
| Telemed1 | <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } ) | 03/08/2011 | 17h:00 | 0 |

Conflictual Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Wait | Assign | Consult | Registr | H-up |
| Wait | 0 | 0 | 0 | 0 | 0 |
| Assign | 0 | 0 | 0 | 0 | 0 |
| Consult | 0 | 0 | 0 | 0 | 0 |
| Registr | 0 | 0 | 0 | 0 | 0 |
| H-up | 0 | 0 | 0 | 0 | 0 |

Synchronous Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Wait | Assign | Consult | Registr | H-up |
| Wait | 1 | 0 | 0 | 0 | 0 |
| Assign | 0 | 1 | 0 | 0 | 0 |
| Consult | 0 | 0 | 1 | 0 | 0 |
| Registr | 0 | 0 | 0 | 1 | 0 |
| H-up | 0 | 0 | 0 | 0 | 1 |

* + - 1. Algorithm running

Lancement de MétaT(ε,telemed1) :

*Structural Conformance*

**Syntax Conformance Result** :

<telemed1>=<registr>{sPt,R};w((<wait>{Pt,sR}\*wε) |> <h-up>{sPt} else (<assign>{sRt, D};w<consult>{Pt, sDt}))

Expression syntaxically correct : Every Operator has two sub-collaborations around it.

**Collaboration Conformance Result** :

Expression Collaborations Conformance correct : Every Collaboration has roles.

*Behavioral Conformance*

**Message Blocking Conformance** :

The interupt operator is the only one that can create Message Blocking Conformance.

**Analyse** : Col1 = <w> ; Col2’ = ε and Col3 = <act>

TR(<w>) = {P} and SR(Col2’) = ε and SR(<act>) = {R}

Thus TR(Col1) != ε and SR(Col2’) U SR(<act>) != ε and SR(Col1) = TR(Col1)

So nothing needed to be changed.

**Deep Blocking Conformance** :

Each matrix are neutral and doesn’t show a real conflictual or synchromous situation. So nothing to be done : Deep Blocking won’t happen.

*Changement Propagation*

Let see the propagation on components :

**Patient** :

**TPatient(ε)** = ε

**TPatient(telemed1)** != ε

So **send** **CreateM(x)** to **MétaUpdateStructure.Create(Patient, TPatient(telemed1))**.

**TPatient(telemed1)** = TP (<registr>) ; ((TP (<w>) |> ( wait(Interr); Interrupted ;= true;) else ε) ; (( receive cim(y) from Receptionist [] TP (<act>) ) | |\* (receive iem(z) from Receptionist; <h-up>; Interr := true; send im(z) to Receptionist))

**Doctor** :

**TDoctor(ε)** = ε

**TDoctor(telemed1)** != ε

So **send** **CreateM(x)** to **MétaUpdateStructure.Create(Doctor, TDoctor(telemed1))**.

**TDoctor(telemed1)** = (TD (<assign>) ; TD (<consult>)) [] receive cim(y) from Receptionist

**Receptionist** :

**TReceptionist(ε)** = ε

**TReceptionist(telemed1)** != ε

So **send** **CreateM(x)** to **MétaUpdateStructure.Create(Doctor, TReceptionist(telemed1))**.

**TReceptionist(telemed1)** = TRecptionist (<registr>) ; (((TDoctor (<w>) |> ( wait(Interr); Interrupted ;= true;) else ε);( receive fim(x, i) from Patient; if i then Interrupted := true; if not Interrupted then TReceptionist (<act>) ) | | (wait(Interrupted); send cim(y) to Doctor and Patient )) | |\* (send iem(z) to Patient; receive im(z) from Patient; Interr := true))

*Conclusion Partielle*

Finallement les composants derivés sont identiques à ceux du cas statique. De ce fait le modèle du professeur Bochmann est conservé. Dans ce qui suit nous allons faire des changement dans le global Behavior. Voilà l’état des matrices à la fin de la dérivation.

Composant-Role-Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Patient | Doctor | Receptionist | State |
| Patient | 1 | 0 | 0 | 1 |
| Doctor | 0 | 1 | 0 | 1 |
| Receptionist | 0 | 0 | 1 | 1 |

Stored-Global-Behaviors

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Expréssion | Informations | | |
| Date | Hour | State |
| Telemed1 | <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } ) | 03/08/2011 | 17h:00 | 1 |

* + 1. Evolving Case : Equipement and Urgence Introduction
       1. Initialisation

Let Patient = P, Equipment = E, Receptionist = R, UrgentAgent = U and Doctor = D.

*Global Behavior Expression*

<telemed2> = <Mesures>{P,E,R};w(<UrgentCase>{P,R,U}[]<UsualCase>{E,D,P,R})

Where :

<Mesures>{P,E,R} = (<HealthConstantsCalculating>{P,E} || <DosageMesures>{E}) ;w <Sending-Brief>{E,R}

<UrgentCase> {P,R,U} = <wait>{U,R} \*w <Alert>{U,P,R}

<UsualCase>{D,E,P,R} = <w>{E,R} |> <Try-Later>{E} else <act>{E,P,D,R}

<w>{E,R} = <wait>{E,R} \*w ε

act>{E,P,D,R} = (<assign>{D,R} ;w <Result-Dosage>{D,R}) ;w (<Retrieve-Dosage>{R,E} ;s <Dosing-Application>{E,P})

*UML Activity Diagram*

*Matrix Generation*

Collaboration-Role-Table

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Recetionist | | | Doctor | | | Equipment | | | Patient | | | UrgentAgent | | |
| SR | TR | PR | SR | TR | PR | SR | TR | PR | SR | TR | PR | SR | TR | PR |
| Mesures | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| UrgentCase | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| UsualCase | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |

Composant-Role-Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Receptionist | Doctor | Equipment | Patient | UrgentAgent | State |
| Receptionist | 1 | 0 | 0 | 0 | 0 | 1 |
| Doctor | 0 | 1 | 0 | 0 | 0 | 1 |
| Equipment | 0 | 0 | 1 | 0 | 0 | 0 |
| Patient | 0 | 0 | 0 | 1 | 0 | 1 |
| UrgentAgent | 0 | 0 | 0 | 0 | 1 | 0 |

Stored-Global-Behaviors

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Expréssion | Informations | | |
| Date | Hour | State |
| Telemed1 | <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } ) | 03/08/2010 | 17h:00 | 1 |
| Telemed2 | <Mesures>{P,E,R};w((<wait>{U,R} \*w <Alert>{U,P,R} )[]( <w>{E,R} |> <Try-Later>{E} else <act>{E,P,D,R})) | 12/02/2011 | 10h:00 | 0 |

* + - 1. Algorithm running

Lancement de MétaT(telemed1,telemed2)

*Structural Conformance*

**Syntax Conformance Result** :

<Mesures>{P,E,R};w((<wait>{U,R} \*w <Alert>{U,P,R} )[]( (<wait>{E,R} \*w ε

) |> <Try-Later>{E} else ((<assign>{D,R} ;w <Result-Dosage>{D,R}) ;w (<Retrieve-Dosage>{R,E} ;s <Dosing-Application>{E,P}))))

Expression syntaxically correct : Every Operator has two sub-collaborations around it.

**Collaboration Conformance Result** :

Expression Collaborations Conformance correct : Every Collaboration has roles.

*Behavioral Conformance*

**Message Blocking Conformance** :

The interrupt, choice and strong sequence can create Message Blocking Conformance.

**Analyse :**

Cas du choix :

IsResponsibleOf Matrix

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Mesures | | UrgentCase | | UsualCase | |
| Mesures | ε | ε | ε | ε | ε | ε |
| UrgentCase | ε | ε | ε | ε | R | R |
| UsualCase | ε | ε | R | R | ε | ε |

So IsResponsible-Of(UrgentCase, UsualCase) = {<R,R>} then nothing to do.

Cas de l’interupt :

**TR((<wait>{E,R} \*w ε))** = {R,E} != ε

**SR(ε) U SR(((<assign>{D,R} ;w <Result-Dosage>{D,R}) ;w (<Retrieve-Dosage>{R,E} ;s <Dosing-Application>{E,P})))** = SR(<assign>{D,R} ;w <Result-Dosage>{D,R}) U {SR(<Retrieve-Dosage>{R,E} ;s <Dosing-Application>{E,P})\{D,R}} = SR(<assign>{D,R}) U {SR(<Result-Dosage>{D,R})\{R,D}} = {R,D} != ε

So nothing to perform too.

Cas de la sequence forte :

**TR(<Retrieve-Dosage>{R,E})** = {R,E} != ε

**SR(<Dosing-Application>{E,P})** = {E} != ε

So Nothing to perform. There won’t be blocking message case here.

Finallement la nouvelle forme ne créera pas de blockage d’attente ou d’envoie de message.

**Deep Blocking Conformance**:

**Changements au niveau des matrices de conflits et de synchronisation**

Conflictual Matrix

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | HealConsCalc | DosageMesures | SendBrief | Wait | Alert | TryLater | Assign | ResultDosage | RetrieeveDosage | DosageApplication |
| HealthConstantsCalculating | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DosageMesures | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SendBrief | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Wait | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Alert | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TryLate | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Assign | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ResultDosage | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| RetrieeveDosage | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DosageApplication | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Synchronous Matrix

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | HealConsCalc | DosageMesures | SendBrief | Wait | Alert | TryLater | Assign | ResultDosage | RetrieeveDosage | DosageApplication |
| HealthConstantsCalculating | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DosageMesures | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SendBrief | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Wait | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Alert | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| TryLate | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Assign | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| ResultDosage | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| RetrieeveDosage | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| DosageApplication | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

We didn’t express constraints so nothing will hapen because of a deep blocking.

*Changement Propagation*

Calculating all Components Behavior :

**Patient** :

**TPatient**.state() = 1

**TPatient(**Telemed1**) =** Tc (<registr>) ; ((TP (<w>) |> ( wait(Interr); Interrupted ;= true;) else ε) ; (( receive cim(y) from Receptionist [] TP (<act>) ) | |\* (receive iem(z) from Receptionist; <h-up>; Interr := true; send im(z) to Receptionist))

**TPatient(**Telemed2**) = TPatient**(<HealthConstantsCalculating>);( **TPatient**(<Alert>)[](((receive cim(y1) from Equipement [] send cim(y2))||\* ε))

**Equipment** :

**TEquipment**.state() = 0

**TEquipment(**Telemed1**) =** ε

**TEquipment(**Telemed2**) !=** ε

So **send** **CreateM(x)** to **MétaUpdateStructure.Create(Equipment, TEquipment(telemed2))**

**TEquipment(**Telemed2**) =** (((TE(<HealthConstantsCalculating>);send cim(y1) to P)||TE(<Dosage-Mesuring>));receive cim(y2) from P ; TE(<Sending-Dosage>)); ε) [](TE(<wait>);receive cim(y3) from R)|>(Wait(Interru); Interrupted:=true;)else ε))||\*(receive im(z1) from R; Interru:=true))

**Doctor** :

**TDoctor**.state() = 1

**TDoctor(**Telemed1**) =** (TD (<assign>) ; TD (<consult>)) [] receive cim(y) from Receptionist

**TDoctor(**Telemed2**) =** (receive cim(y4) from R)[]((receive fim(x1,i1) fromR; if i1 then Interrupted:=true; if not Interrupted then send cim(y5)||\* ε to R)

**UrgentAgent** :

**TUrgentAgent**.state() = 0

**TUrgentAgent(**Telemed1**) =** ε

**TUrgentAgent(**Telemed2**) !=** ε

So **send** **CreateM(x)** to **MétaUpdateStructure.Create(UrgentAgent, TUrgentAgent(telemed2))**

**TUrgentAgent(**Telemed2**) =** ((TU(<wait>);receive cim(y6) from R;TU(<Alert>);send cim(y7) to R)[](receive cim(y8) from R)

**Receptionist** :

**TReceptionist**.state() = 1

**TReceptionist(**Telemed1**) =** TRecptionist (<registr>) ; (((TDoctor (<w>) |> ( wait(Interr); Interrupted ;= true;) else ε);( receive fim(x, i) from Patient; if i then Interrupted := true; if not Interrupted then TReceptionist (<act>) ) | | (wait(Interrupted); send cim(y) to Doctor and Patient )) | |\* (send iem(z) to Patient; receive im(z) from Patient; Interr := true))

**TReceptionist(**Telemed2**) =** TR(<sending-Brief>);(((TR(<wait>);TR(<Alert>);send cim(y6) to U);send cim(y4) to D and cim(y6) to U)[](((TR(<wait>)|>Wait(Interru); Interrupted:=true;else ε)||\*(I-Enabled :=true ;Wait(I-Enabled) ;<Try-Later> ;Interru=true ;send im(z1) to E)) send cim(y4) to D and fim(x1,i1) to U)

*Conclusion Partielle*

Le système vire alors vers un nouvel état suivant la nouvelle expréssion. Les matrices de composants-role et de Stored-Global Behavior aussi.

Composant-Role-Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Receptionist | Doctor | Equipment | Patient | UrgentAgent | State |
| Receptionist | 1 | 0 | 0 | 0 | 0 | 1 |
| Doctor | 0 | 1 | 0 | 0 | 0 | 1 |
| Equipment | 0 | 0 | 1 | 0 | 0 | 1 |
| Patient | 0 | 0 | 0 | 1 | 0 | 1 |
| UrgentAgent | 0 | 0 | 0 | 0 | 1 | 1 |

Stored-Global-Behaviors

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Expréssion | Informations | | |
| Date | Hour | State |
| Telemed1 | <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } ) | 03/08/2010 | 17h:00 | 1 |
| Telemed2 | <Mesures>{P,E,R};w((<wait>{U,R} \*w <Alert>{U,P,R} )[]( <w>{E,R} |> <Try-Later>{E} else <act>{E,P,D,R})) | 12/02/2011 | 10h:00 | 1 |

<HealthCon> = send h1 from E to P ; receiveh2 from P to E

<Sending-Brief> = send sb1 from E to R

<Wait> = send w1 from R to E; send w1 from R to U

<Alert> = send al1 from E to R; send al2 from R to U

<Assign> = send as1 from R to D; receive as2 from D to R

<Result-Dos> = send rdo1 from D to R

<Retrieve-D> = send dos1 from R to E

<Dosing-App> = send dap1 from E to P; receive dap2 from P to E

Patient

Equipment

Receptionist

Doctor

UrgentAgent

h1

h2

dap1

al3

w1

im

dos1

cim[]ass1

cim[]al2

rdo1

dap2

cimm

cim

cim

cim

cim

w2

fim

sb1[]al1

Pour illustrer le cas des suppressions de composants et de leur mise à jour nous avons considéré, dans ce qui suit, le fait que le système retourne à sa configuration initiale.

* + 1. Returning to initial case : Equipment Deficient Impact
       1. Initialisation

*Global Behavior Expression*

<w>{P, sRt} = <wait>{Pt, sR} \*w ε

<act>{Pt, sRt, Dt} = <assign>{sRt, D} ;w <consult>{Pt, sDt}

<telemed1> = <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } )

*UML Activity Diagram*

*Matrix Generation*

Collaboration-Role-Table

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Receptionist | | | Doctor | | | Equipement | | | Patient | | | UrgentAgent | | |
| SR | TR | PR | SR | TR | PR | SR | TR | PR | SR | TR | PR | SR | TR | PR |
| registr | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| w | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| act | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| h-up | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

Composant-Role-Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Receptionist | Doctor | Equipment | Patient | UrgentAgent | State |
| Receptionist | 1 | 0 | 0 | 0 | 0 | 1 |
| Doctor | 0 | 1 | 0 | 0 | 0 | 1 |
| Equipment | 0 | 0 | 1 | 0 | 0 | 1 |
| Patient | 0 | 0 | 0 | 1 | 0 | 1 |
| UrgentAgent | 0 | 0 | 0 | 0 | 1 | 1 |

Stored-Global-Behaviors

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Expréssion | Informations | | |
| Date | Hour | State |
| Telemed1 | <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } ) | 03/08/2010 | 17h:00 | 1 |
| Telemed2 | <Mesures>{P,E,R};w((<wait>{U,R} \*w <Alert>{U,P,R} )[]( <w>{E,R} |> <Try-Later>{E} else <act>{E,P,D,R})) | 12/02/2011 | 10h:00 | 1 |
| Telemed3 | <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } ) | 03/08/2011 | 07h:00 | 0 |

* + - 1. Algorithm running

Lancement de MétaT(telemed2,telemed3)

*Structural Conformance*

Etant donner qu’il s’agit de la même expréssion que telemed vous conviendrez avec nous que la structural conformance est correcte et ce parceque cette expréssion est la première a avoir été dérivé.

*Behavioral Conformance*

De même, la Behavioral Conformance est aussi correcte du fait du tableau des Stored Behavior où le state est 1 ce qui signifit que cette expréssion est déjas correct.

*Changement Propaggation*

Calculating all Components Behavior :

**Patient** :

**TPatient**.state() = 1

**TPatient(**Telemed3**) =** Tc (<registr>) ; ((TP (<w>) |> ( wait(Interr); Interrupted ;= true;) else ε) ; (( receive cim(y) from Receptionist [] TP (<act>) ) | |\* (receive iem(z) from Receptionist; <h-up>; Interr := true; send im(z) to Receptionist))

**TPatient(**Telemed2**) = TPatient**(<HealthConstantsCalculating>);( **TPatient**(<Alert>)[](((receive cim(y1) from Equipement [] send cim(y2))||\* ε))

On a **TPatient(**Telemed3**) ! =** ε **et TPatient(**Telemed2**) !=** ε

**So send UpdateM(x1)** to **UpdateBehavior(Patient, TPatient(**Telemed3**))**

**Equipment** :

**TEquipment**.state() = 1

**TEquipment(**Telemed3**) =** ε

**TEquipment(**Telemed2**) =** (((TE(<HealthConstantsCalculating>);send cim(y1) to P)||TE(<Dosage-Mesuring>));receive cim(y2) from P ; TE(<Sending-Dosage>)); ε) [](TE(<wait>);receive cim(y3) from R)|>(Wait(Interru); Interrupted:=true;)else ε))||\*(receive im(z1) from R; Interru:=true))

**TEquipment(**Telemed2**) !=** ε

So send **DeleteM(x2)** to **UpdateStructure.Delete(Equipment)**

**Doctor** :

**TDoctor**.state() = 1

**TDoctor(**Telemed3**) =** (TD (<assign>) ; TD (<consult>)) [] receive cim(y) from Receptionist

**TDoctor(**Telemed2**) =** (receive cim(y4) from R)[]((receive fim(x1,i1) fromR; if i1 then Interrupted:=true; if not Interrupted then send cim(y5)||\* ε to R)

On a **TDoctor(**Telemed3**) !=** ε **et TDoctor(**Telemed2**) !=** ε

**So send UpdateM(x)** to **UpdateBehavior(Doctor, TDoctor(**Telemed3**))**

**UrgentAgent** :

**TUrgentAgent**.state() = 1

**TUrgentAgent(**Telemed3**) =** ε

**TUrgentAgent(**Telemed2**) =** ((TU(<wait>);receive cim(y6) from R;TU(<Alert>);send cim(y7) to R)[](receive cim(y8) from R)

**TUrgentAgent(**Telemed2**) !=** ε

So **send DeleteM(x)** to **UpdateStructure.Delete(UrgentAgent)**

**Receptionist** :

**TReceptionist**.state() = 1

**TReceptionist(**Telemed3**) =** TRecptionist (<registr>) ; (((TDoctor (<w>) |> ( wait(Interr); Interrupted ;= true;) else ε);( receive fim(x, i) from Patient; if i then Interrupted := true; if not Interrupted then TReceptionist (<act>) ) | | (wait(Interrupted); send cim(y) to Doctor and Patient )) | |\* (send iem(z) to Patient; receive im(z) from Patient; Interr := true))

**TReceptionist(**Telemed2**) =** TR(<sending-Brief>);(((TR(<wait>);TR(<Alert>);send cim(y6) to U);send cim(y4) to D and cim(y6) to U)[](((TR(<wait>)|>Wait(Interru); Interrupted:=true;else ε)||\*(I-Enabled :=true ;Wait(I-Enabled) ;<Try-Later> ;Interru=true ;send im(z1) to E)) send cim(y4) to D and fim(x1,i1) to U)

On a **TReceptionist(**Telemed3**) !=** ε **et TReceptionist(**Telemed2**) !=** ε

So **send UpdateM(x)** to **UpdateBehavior(Receptionist, TReceptionist(**Telemed3**))**

*Conclusion Partielle*

Le nouvel état du système a étrainé la suppression de certains composants comme vous le verez dans les tableaux suivant. L’expréssion telemed3 est donc dérivée avec succès et deveint le nouvel état stable dans lequel se trouvera le système.

Composant-Role-Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Receptionist | Doctor | Equipment | Patient | UrgentAgent | State |
| Receptionist | 1 | 0 | 0 | 0 | 0 | 1 |
| Doctor | 0 | 1 | 0 | 0 | 0 | 1 |
| Equipment | 0 | 0 | 1 | 0 | 0 | 0 |
| Patient | 0 | 0 | 0 | 1 | 0 | 1 |
| UrgentAgent | 0 | 0 | 0 | 0 | 1 | 0 |

Stored-Global-Behaviors

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Expréssion | Informations | | |
| Date | Hour | State |
| Telemed1 | <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } ) | 03/08/2010 | 17h:00 | 1 |
| Telemed2 | <Mesures>{P,E,R};w((<wait>{U,R} \*w <Alert>{U,P,R} )[]( <w>{E,R} |> <Try-Later>{E} else <act>{E,P,D,R})) | 12/02/2011 | 10h:00 | 1 |
| Telemed3 | <registr>{sPt, R} ;w (<w>{P, sRt } |> <h-up>{sPt} else <act>{ Pt, sRt, Dt } ) | 03/08/2011 | 07h:00 | 1 |