An MDE Approach to Derive System component Behavior

Ahmed Harbouche[1], Mohammed Erradi[2] and Aicha Mokhtari[3]

[1]*LME Lab, Hassiba BenBouali University, Chlef, Algeria*
[2]*SIME Research Lab, ENSIAS, Mohammed V Souissi University, Rabat Morocco*
[3]*LRIA Lab, Houari Boumedienne University, Algiers, Algeria*
A.harbouche@univ-chlef.dz, erradi@ensias.ma, Aissani_Mokhtari@yahoo.fr

Abstract

In this paper we are interested in the early development phases of a distributed software system, especially in the case of obtaining a system design from its global requirement specification. This is an important step in the development of complex systems and their automated derivation. The Model-Driven Engineering (MDE) approach allows the designers to model their systems at different abstraction levels. Such approach provides automatic model transformations to incrementally refine abstract models into concrete ones. This paper presents a MDE approach to derive the behavior of a given system component from its global requirements. The requirements are specified using UML activity diagram extended with collaborations. The derived behavior of the components is in the form of distributed UML state machines. The suggested approach is based on the definition of an appropriate source meta-model (requirements meta-model) and the definition of a target design meta-model. A set of model transformation rules has been designed to govern the derivation process. A real application of telediagnosis in neuroscience has been developed using this approach.

Keywords: *MDE, Behavior, Derivation, Model to model transformation*

1. Introduction

Distributed software systems development is an increasingly complex task. Usually the global behavior of distributed systems is not achieved by a single component but by a set of collaborative components. Such global behavior can be decomposed into partial behaviors performed by different system components. A manual decomposition of the behavior can lead to design errors. Therefore an automatic transformation approach is needed to derive the behavior of these components from the system global requirements. The use of a software engineering approach to support such derivation process may lead to a highly platform independent designs and promotes the reuse of software artifacts where a derived behavior model can be reused on different platforms.

The Model-Driven Engineering (MDE) approach provides models to define the requirements, design, deployment and implementation of a given system. Each model describes the system at a given level of abstraction. Therefore model transformations are needed between the different levels of abstraction. Such model transformations are useful to obtain final application code from the system requirements model through a refinement process. Each level of abstraction is described using a specific meta-model. A mapping between these meta-models is needed to define the refinement process. Thus, applying a MDE approach requires the appropriate meta-models and the corresponding model transformations. Many researches on MDE are focusing on transforming the design to deployment models [1-7]. However, less attention has been given to the transformations of the requirements to the design models especially in the case of distributed applications where

specific properties, such as messages exchange and collaborations between the distributed components, need to be addressed.

This paper presents a MDE approach to derive the behavior of the system components by transforming the system global requirements model to the design model. The system global requirements model describes the functional behavior of a given system in an abstract way. The design model represents the local behavior of each component. The appropriate meta-models have been defined at each level of abstraction with the corresponding model transformations. The proposed approach allows designers to describe their system using UML activity diagram extended with collaborations. This requirements model is then automatically transformed to the behavior of the components. A set of rules is needed to govern the model transformations during the derivation process. As we are in a distributed context, the derived behaviors of the components need also to be synchronized. The collaboration of these behaviors should satisfy the initial requirements model.

This paper is structured as follows. Section 2 reviews the related works. Section 3 presents the proposed derivation process overview and describes the basic meta-models. Section 4 presents the transformation rules. Section 5 presents a case study. Section 6 presents the implementation of the system design derivation process. Section 7 concludes this paper.

2. Related Works

Our work is related to Model-Driven Engineering (MDE), behavior derivation and requirements specification.

The MDE specifies the requirements, design, deployment and implementation code models to describe systems. The transformation of the requirements models to the design models is the first step to quality design of a complex system. The works [8-10] provide a model transformation of the CIM (Computational Independent Model) to PIM (Platform Independent Model). While STREAM (A Strategy for Transition between Requirements Models and Architectural Models) presented in [11] generates an architectural model described in Acme architectural description language from i* requirements model. Although these proposals are similar to ours in the sense of using transformations between requirements and design models, they do not explore the case of distributed applications where specific properties, such as message exchange and collaborations between the distributed components, need to be addressed. We also provide the behavior derivation during the model transformations.

The behavior derivation needs a transformation approach to automatically derive the behavior of the components from the system global requirements. Bochmann [12] suggests an algorithm for the derivation of behavior based on behavior expressions as basic constructs for the system global requirements. The expressions of collaborations describe the sequential, choice, repetition and parallel structures. While they could not include the conditions (guards) embedded in the choice structures and the repetition structure. In addition, the derived behaviors are in the form of expressions. These expressions cannot be verified and validated automatically. Therefore our approach allows the behavior derivation based on model transformations. The models provide an abstract representation of collaborations with a clear understanding of the underlying system. UML activity diagrams extended with collaborations are used to describe the system requirements. They allow representing concepts that could not be described using expressions, as done in [12]. This is the case of the conditions and control nodes. The derived behaviors of the system components are UML state machines which could be used for an automatic code generation. The derived state machines models can be verified and validated using existent tools such as the SPIN model checker [13]. In

addition, our approach provides a way to generate platform independent models which could be implemented on more than one platform.

The requirements specification describes the system global behavior. To express the system global behavior, UML interaction and communications diagrams [14] are generally used to describe collaborative behavior. However, they are expressed in terms of message exchanges between the various components cooperating at the preliminary stages of development where it is not necessary to have too many details. To represent the system global behavior, we suggest the use of UML collaborations as the main blocks of activities for the construction of the requirements models. A given collaboration describes the structure of collaborating elements (roles), each performing a specialized function which collectively accomplishes a desired functionality. The collaborations are very appropriate to model the requirements because they provide a structural framework for such requirements which embody both the behavior of each role and the interactions between the roles. This allows describing a given system global requirements in an abstract level.

3. The derivation process overview and basic meta-models

Within the MDE context, many software architects understand and argue that the requirements level and its transformation to the design level is the first step to quality design of a complex system [8-10]. In this direction, we define a model-driven approach to derive the behavior of the system components. This approach will allow developers to build more flexible and reusable designs. The main goal of this approach is to define the derivation process. This goal was achieved by:

1. The definition of the requirements meta-model (source meta-model) which enables the description of the system global behavior at a very high level of abstraction.
2. The description of the design meta-model (target meta-model) which allows to describe the local behavior of each system component.
3. The definition of the “model to model” transformation which maps the concepts from the requirements meta-model to those of the design meta-model. These transformation rules will govern the derivation process.

3.1. The basic meta-models

The basic meta-models in our approach are the requirements meta-model and the design meta-model. The requirements meta-model is defined by an UML activity diagram extended with *collaborations*. The design meta-model is the UML *state machine* meta-model.

3.1.1. The requirements meta-model: The definition of the requirements meta-model is aimed to describe the system global behavior at a high level of abstraction. At this stage, no design concepts or information about the final target platform are taken into account. Models at this level must provide a clear picture of the system global behavior and the collaborations between its components.

The proposed requirements meta-model (Figure 1) has been designed to provide the appropriate concepts and their relationships used to describe a collaborative system. Such requirements meta-model is considered as the source meta-model of the derivation process. It defines the activity diagram meta-model with its main classes and associations while considering collaborations as the basic activities. An activity diagram consists in several

ActivityNode and *ActivityEdge*. An *ActivityEdge* class allows specifying the control flow (connections) between *ActivityNode* classes. The *ActivityNode* class specifies the different sequencing operators of the activities defined by the class *ControlNode* and the activities (*Action*). The activities are defined as *Collaborations* involving multiple roles. The *Collaboration* in the requirements meta-model may consist in one or two collaborations or sub-collaborations. A *Sub-collaboration* consists in some actions accomplished by the collaborating roles. The *Roles* represent the different system components. UML activity diagrams are suitable to represent choreography of collaborations and sub-collaborations. These collaborations are the basic activities in a composite collaboration describing the system global behavior. UML activity diagrams can express sequential behavior, alternative behavior, competing behavior (parallel composition), repetitive behavior, as well as interruptions. At the models level, the system global behavior is defined as a composition of activity diagrams and collaborations.

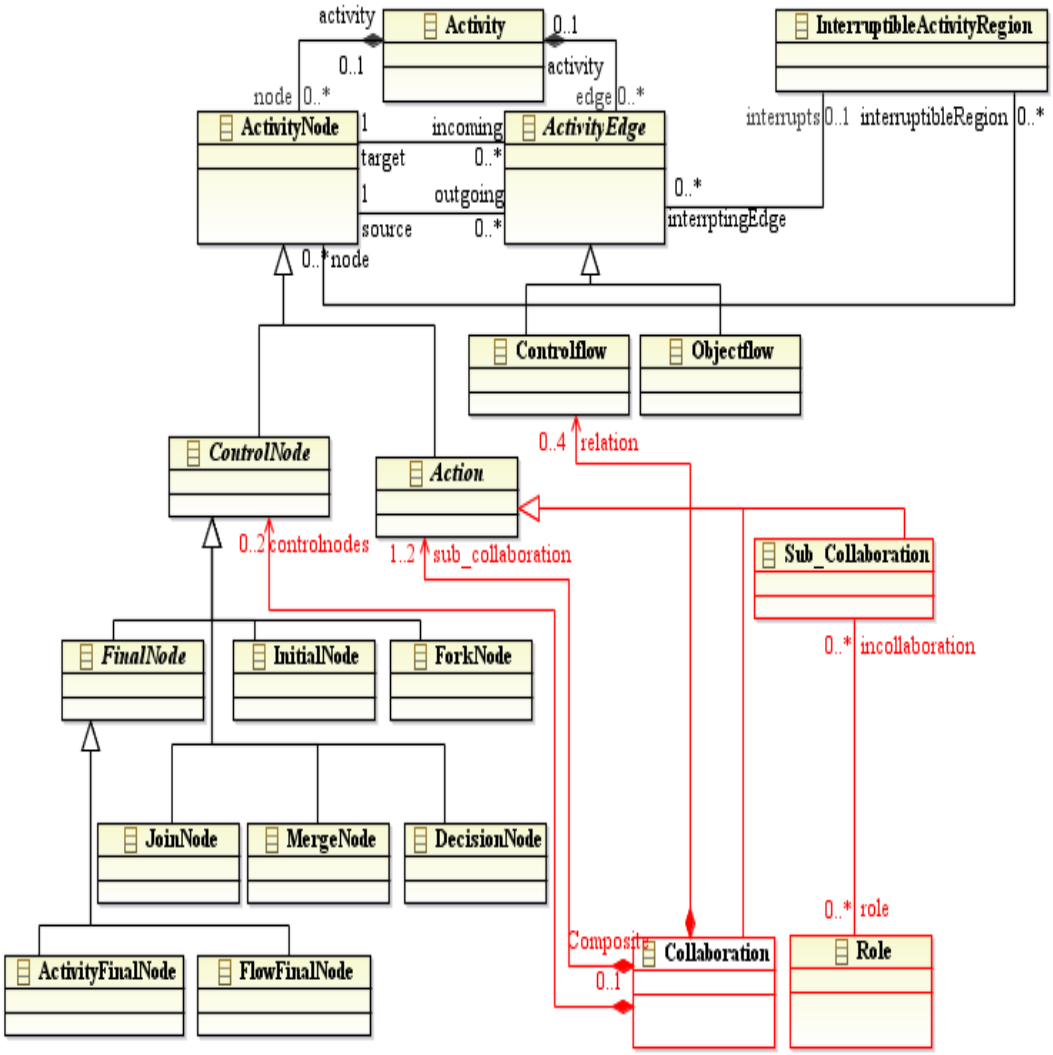

Figure 1. The requirements meta-model

3.1.2. The design meta-model: The behavior of the various components or roles of a given system could be modeled using UML activity diagrams or UML state machines. The UML state machine meta-model [14] was selected as the design meta-model. It is a platform independent modeling language. Thus, models automatically generated at this level from the requirements ones provide a more detailed description of the behavior of the various components. The transformations of these models to code could be performed and automated. Producing code is out of the scope of this work.

3.2. The model to model transformation

The model to model transformation describes how the requirements models are automatically transformed into UML state machines. It governs the derivation process by a set of transformation rules. A rule consists in transforming a concept outlined in the requirements meta-model to a corresponding one in the target design meta-model. For this purpose, we define the function named *Transform (Requirement_Concept, Statemachine_Concept, Messages)*. This function performs some relatively complex mappings, although some of them are also quite direct. Some of these transformations are outlined as follows:

- Each *Controlflow* concept is mapped to a *Transition* concept.
- Each *ControlNode* concept is mapped to a *ControlState* concept.
- Each *Sub-collaboration* concept is mapped to a *CompositeState*. This composite state will hold the actions of the concerned role after transformation.
- Each *Collaboration* concept is mapped to a *CompositeState*. Then the function *Transform* will trigger the transformation of its sub-collaborations.

The *Messages* parameter in this function represents the coordination messages that will be included in the generated composite state to ensure the global coordination among the derived system roles. The result of applying the designed model to model transformation to the initial global requirements is UML state machine platform-independent models. These models describe the behavior of the system components. The corresponding transformation rules will be described in section 4.

3.3. The derivation process algorithm

The derivation process algorithm (Algorithm 1) determines, for a given role, the appropriate rule to be applied for each concept represented in the source model (Ms) and it generates the corresponding concept in the target design model (Mt). The *Collaboration* concept is transformed into a composite state. This composite state may hide an UML state machine which is the specification of the actions of such given role. The algorithm will also generate the needed transitions to interconnect the generated states in the composite state. In addition to the role actions, such composite state may also contain the corresponding coordination messages for realizing the synchronization between the derived behaviors. For each *Controlflow* concept, the algorithm will apply the appropriate rule in order to generate the needed transitions to connect the generated composite state to its predecessor and successor states. Similarly, the *ControlNode* concept is transformed into a *ControlState* and connected by the appropriate transitions.

```
Init r = Role_name;
for all Concept= collaboration C in Ms do
   Generate a composite State S in Mt conform to MMt;
   S.Name= C.Name;   Generate (Initial State) in S;
   if type_collaboration C = sub-collaboration then
      for each action a in C do Transform(a, a') endfor;
   else
      for each C' in C do Transform(C', S', Messages) endfor;
   endif;
   if Messages≠ Nil then Integrate the appropriate coordination messages in S
endif;
   Generate the transitions to connect the states in the composite state S;
   Generate (Final State) in S;
endfor;

for all Concept = Controlflow C or ControlNode C in Ms do
   Transform (controlflow, Transition);
   Transform (controlNode, ControlState);
endfor;
```

Algorithm 1. The derivation process algorithm

4. Transformation rules

The definition of the transformation rules, to govern the derivation process, requires the identification of various relationships between the requirements and the target design meta-models. To perform the derivation, we identify several cases of choreography expressed in the requirements meta-model: sequential, alternative (choice composition), competition (parallel composition) and repetition behaviors. At the level of collaboration, the starting roles (SR) and terminating roles (TR) are distinguished as in [12] (Figure 2).

Figure 2. Structure of the collaboration

Definition 1: A *starting role* is a role that accomplishes an initial action in a collaboration or in one of its sub-collaborations.

Definition 2: A *terminating* role is a role that accomplishes a final action in a collaboration or in one of its sub-collaborations.

The sets of starting (SR), terminating (TR) and participating roles (PR) are calculated using the same techniques as in [12]. These sets are calculated for a given collaboration depending on the sequencing operators used in the activity diagram. Two types of sequencing are distinguished [15-16]: Strong and Weak sequencing.

Definition 3: *Strong sequencing* (Figure 3) implies that all sub-activities of A1 are finished before an activity A2 can begin.

Definition 4: *Weak sequencing* (Figure 4) specifies that an activity A2 will be executed after another activity A1. Weak sequencing provides only a local order of activities for each system component and does not imply a global order.

In addition, two coordination messages are introduced in the derivation process to ensure the synchronization between the derived system roles. We use the same kind of coordination messages as introduced in [12]. Each coordinating message contains the parameters: a) Source role (*Sr*), b) Destination role (*Dr*) and c) name of state it-belong to (*St*). The coordination messages are:

1. Flow message for coordinating strong sequencing, named *Flowm(Sr, Dr, St)*.
2. Choice indication message for propagating the choice to a role that doesn't participate in the selected alternative in the choice composition structure, named *Choicem(Sr, Dr, St)*.

We have defined the transformation rules for the different cases of choreography expressed in the requirements meta-model. In the following, we express the rules that make possible to derive the behavior of the different roles involved in a system global requirements specification. Each rule performs the appropriate model to model transformation for a role *r* participating in a collaboration C_i. The transformation of the collaboration concept triggers the transformation of its sub-collaborations in the case where it is composed. This property requires the definition of recursive transformation rules.

4.1. Strong sequencing between two collaborations

The flow message *Flowm(Sr, Dr, St)* is used for synchronizing the strong sequencing between the derived behavior of the roles.

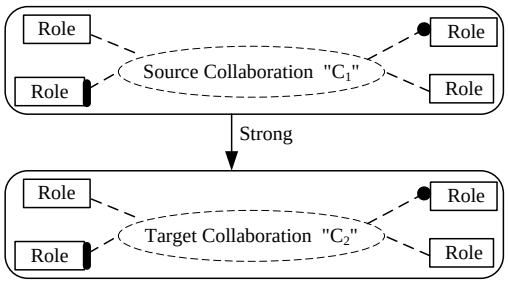

Figure 3. Strong sequencing between two collaborations

Rule 1: The source collaboration C_1 is transformed into the composite state S_1, for a terminating role *r* in C_1. This state S_1 will hold the actions performed by the role *r* after transformation and include the actions of sending the coordination messages *Flowm*. The coordination message *Flowm* is sent by the role *r* to the starting roles of the target collaboration C_2 (except to itself if it is a member of this set of roles).

$$\text{If } r \in TR(C_1) \text{ then } Transform(C_1, S_1, Send(Flowm(r, r', S_2))) \qquad \forall r' \in (SR(C_2)-r);$$

Rule 2: The target collaboration C_2 is transformed into the composite state S_2, for a starting role *r* in C_2. This state consists in the actions of receiving the coordination messages *Flowm* from the terminating roles of C_1 (except from itself) and includes the actions performed by *r* after transformation.

If $r \in SR(C_2)$ *then Transform*$(C_2, S_2, Receive(Flowm(r', r, S_2)))$ $\forall r' \in (TR(C_1)-r)$;

Rule 3: The collaboration concept is transformed into a composite state, for a participant role *r*, not terminating in the source collaboration or not starting in the target collaboration. This state holds the actions performed by the role *r*.

If $r \in (PR(C_1) - TR(C_1))$ *or* $r \in (PR(C_2) - SR(C_2))$ *then Transform*(C_i, S_i) $\forall i=1,2$;

Rule 4: The controlflow concept is transformed into a transition, for a participant role *r* in the source and the target collaborations. This transition connects the states S_1 and S_2 obtained from the transformation of C_1 and C_2.

If $r \in PR(C_1)$ *and* $r \in PR(C_2)$ *then Transform(Controlflow, Transition);*

4.2. Weak sequencing between two collaborations

In this case, no coordination message is used. To transform the controlflow concept, Rule 4 is applied.

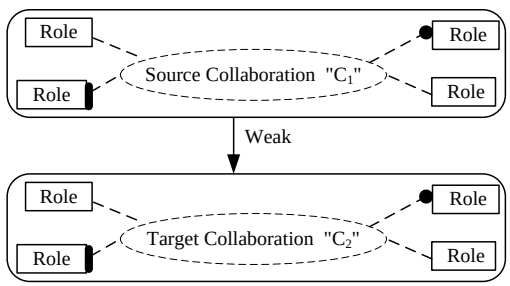

Figure 4. Weak sequencing between two collaborations

Rule 5: The collaboration C_i is transformed into the composite state S_i, for a participant role *r* in the collaboration C_i. This state holds the actions performed by *r*.

If $r \in PR(C_i)$ *then Transform*(C_i, S_i) $\forall i=1,2$;

4. 3. Choice between two collaborations

We consider that the decision is made locally in a given role. Such role is the starting role in the collaboration choice structure (it is the starting role in both collaborations of the choice composition) (Figure 5). The Choice indication message *Choicem(Sr, Dr, St)* is used for propagating the choice to a role that doesn't participate in the selected alternative of the choice composition structure.

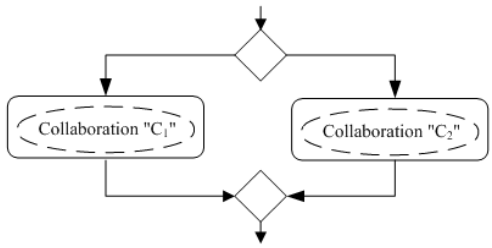

Figure 5. Choice between two collaborations

Rule 6: The collaboration concept C_i is transformed into the composite state S_i, for a participant role r in a choice composition and responsible for the choice. This composite state S_i will hold the actions of the concerned role after transformation. As the role is responsible for the choice, it must send in parallel a coordination message *Choicem* to all roles that do not participate in the selected collaboration C_i alternative but are members of the other alternative $C_{i'}$.

$$\text{If } r \in SR(C_i) \text{ then } Transform(C_i, S_i, Send(Choicem(r, r', S_i))) \quad \forall r' \in (PR(C_{i'}) - PR(C_i)) \text{ and } \forall i, i'=1,2 \text{ and } i \neq i';$$

Rule 7: The collaboration concept C_i is transformed into the composite state S_i, for a participant role r in a choice composition and not responsible for the choice. This composite state S_i will hold the actions of the concerned role.

$$\text{If } r \in PR(C_i) \text{ and } r \notin SR(C_i) \text{ then } Transform(C_i, S_i) \quad \forall i=1,2;$$

Rule 8: For a role r involved in one of the two collaborations of a choice structure, the collaboration concept is transformed into a composite state corresponding to the collaboration in which it doesn't participate. This composite state consists in an action for receiving the coordination message *Choicem* from the starting role of the choice structure.

$$\text{If } r \in (PR(C_i) - PR(C_{i'})) \quad \forall i, i'=1,2 \text{ and } i \neq i' \text{ then } Transform(C_{i'}, S_{i'}, Receive(Choicem(r', r, S_i))) \quad \forall r' \in SR(C_{i'});$$

Rule 9: The controlflow concept is transformed into a transition, for a participant role r in a collaboration C_i. This transition connects the states S_1 and S_2 obtained from the transformation of C_1 and C_2 to the control states. When a guard is associated to the controlflow, the guard is associated to the generated transition in the derived model of the starting role.

$$\text{If } r \in PR(C_i) \text{ then } Transform\ (Controlflow, Transition) \quad \forall i=1,2;$$

Rule 10: The DecisionNode concept is transformed into a ChoiceState, for a participant role r in a collaboration C_i. This state is connected to the composite states S_1 and S_2 obtained from the transformation of C_1 and C_2, by the transitions obtained by application of Rule 9.

$$\text{If } r \in PR(C_i) \text{ then } Transform(DecisionNode, ChoiceState) \quad \forall i=1,2;$$

Rule 11: The MergeNode concept is transformed into a JunctionState, for a participant role r in a collaboration C_i. This state is connected to the composite states S_1 and S_2 obtained from the transformation of C_1 and C_2, by the transitions obtained by application of Rule 9.

$$\text{If } r \in PR(C_i) \text{ then } Transform(MergeNode, JunctionState) \quad \forall i=1,2;$$

4.4. Example

An example is shown in this section to illustrate the derivation process. The system global requirement (Figure 6) is described by a composite collaboration. This collaboration is a sequence between the sub-collaboration C_1 and a composite collaboration named C_2. The collaboration C_2 defines a choice between the sub-collaborations C_3 and C_4. We assume that each sub-collaboration consists in a single action.

The Table 1 shows the calculated sets of the starting, terminating and participants roles at each collaboration involved in the system global behavior.

Table 1. The sets of starting, terminating and participants roles

Choreography case	Starting Roles (SR)	Terminating Roles (TR)	Participants Roles (PR)
Sub-collaboration "C_1"	{r1}	{r1}	{r1, r2, r3}
Sub-collaboration "C_3"	{r1}	{r1, r4}	{r1, r2, r3, r4}
Sub-collaboration "C_4"	{r1}	{r3}	{r1, r3}
Collaboration "C_2"	{r1}	{r1, r3, r4}	{r1, r2, r3, r4}

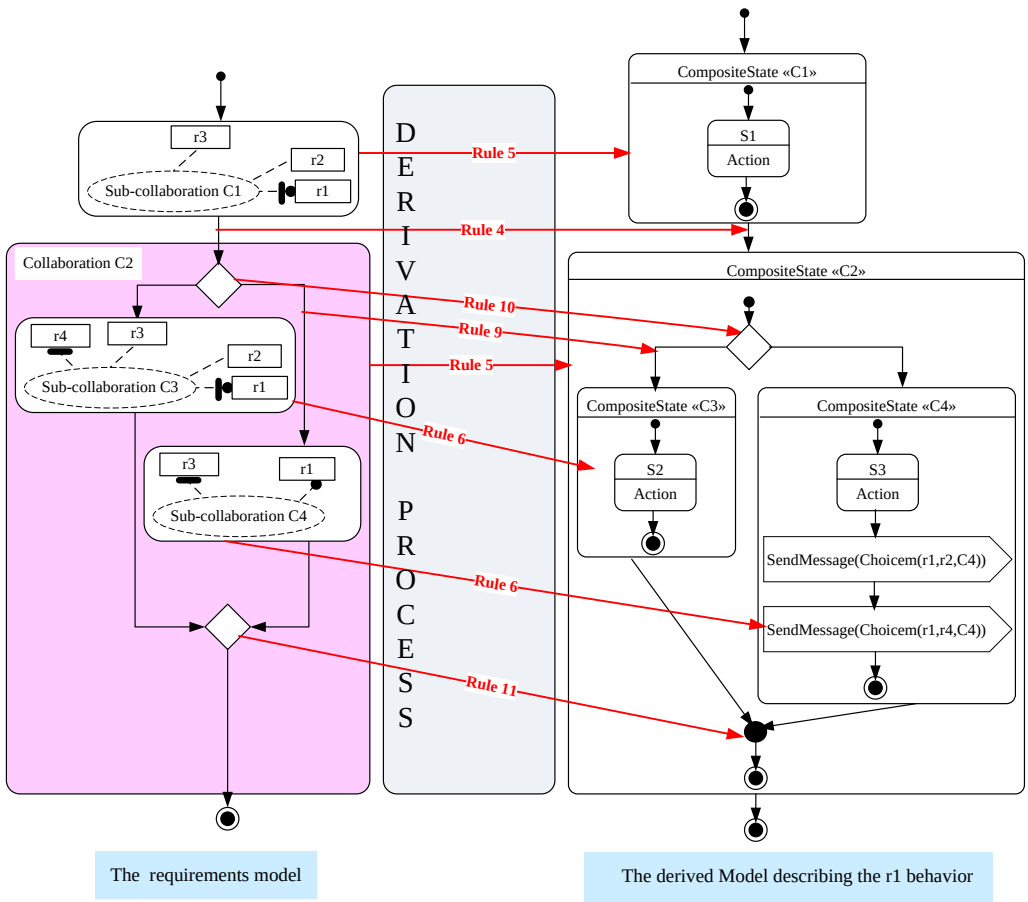

Figure 6. Example of the model to model transformation (derivation process)

We consider in this example the transformation rules that allow deriving the behavior of the role *r1*. The model consists in a weak sequence between C_1 and C_2, and the role *r1* participates in the two collaborations. Therefore the derivation process triggers the rules 4 and 5. The rule 5 realizes the transformation of the actions that it performs at the sub-collaboration C_1 and triggers the transformation of the collaboration C_2. The rule 4 performs the transformation of the control flow in a transition that connects the composite states resulting from the transformation of C_1 and C_2. The collaboration C_2 is a choice composition between the sub-collaborations C_3 and C_4, and the role *r1* is responsible for the choice (it is a starting role in the two sub-collaborations). Therefore the derivation process triggers the rules

6, 9, 10 and 11. The rule 6 performs the transformation of the actions that it performs at each sub-collaboration. On the other hand, it must inform the roles *r2* and *r4* not participating in the collaboration C_4 by sending a coordination message *Choicem* for indicating the choice of C_4. The rules 9, 10 and 11 perform the transformation of the control flow and the control nodes. The derivation process generates a state machine which describes the *r1* behavior (Figure 6).

4.5. While loop with strong sequencing (local choice)

There is strong sequencing between the end of C_1 and the choice of executing C_1 again or terminating the loop with C_2 (Figure 7). The flow message *Flowm(Sr, Dr, St)* is used for synchronizing the strong sequencing between the derived behaviors of the roles. The Choice indication message *Choicem(Sr, Dr, St)* is used for propagating the choice to the roles that participate in the while loop structure (the collaboration C_1) and don't participate in the collaboration C_2.

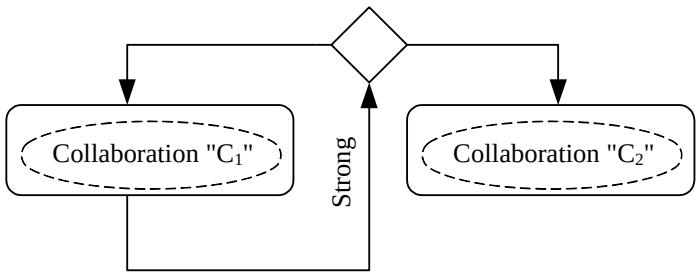

Figure 7. While loop with strong sequencing

Rule 12: The collaboration C_1 is transformed into the composite state S_1, for a terminating role *r* in the collaboration C_1. This state will hold the actions performed by *r* and include the actions of sending the coordination messages *Flowm*. The coordination message *Flowm* is sent by the role *r* to the starting roles of the collaboration C_1 (except to itself if it is a member of this set of roles).

$If\ r \in TR(C_1)\ then\ Transform(C_1, S_1, Send(Flowm(r, r', S_1))) \qquad \forall r' \in (SR(C_1)-r);$

Rule 13: The collaboration C_1 is transformed into the composite state S_1, for a starting role *r* in C_1. This state consists in the actions of receiving the coordination messages *Flowm* from the terminating roles of C_1 (except from itself) and includes the actions performed by *r*.

$If\ r \in SR(C_1)\ then\ Transform(C_1, S_1, Receive(Flowm(r',r,S_1))) \qquad \forall r' \in (TR(C_1)-r);$

Rule 14: The collaboration concept C_1 is transformed into the composite state S_1, for a participant role *r*, not terminating and not starting in C_1. This state holds the actions performed by *r*.

$If\ r \in PR(C_1)\ and\ (r \notin SR(C_1)\ and\ r \notin TR(C_1))\ then\ Transform(C_1, S_1);$

Rule 15: The collaboration concept C_2 is transformed into the composite state S_2, for a starting role *r* in the collaboration C_2. This composite state S_2 will hold the actions performed by *r* after transformation and include the actions of sending the coordination messages *Choicem* to all roles that do not participate in the selected collaboration C_2 but are members of the collaboration C_1.

If $r \in SR(C_2)$ *then Transform*$(C_2, S_2, Send(Choicem(r, r', S_2)))$ $\forall r' \in (PR(C_1) - PR(C_2))$;

Rule 16: The collaboration C_2 is transformed into the composite state S_2, for a participant role *r* and not a starting role in C_2. This state will hold the actions of the concerned role.

If $r \in PR(C_2)$ *and* $r \notin SR(C_2)$ *then Transform*(C_2, S_2);

Rule 17: The collaboration C_2 is transformed into the composite state S_2, for a participant role *r* in C_1 but not participating in C_2. This state will hold an action for receiving the coordination message *Choicem* from the starting role of C_2.

If $r \in (PR(C_1) - PR(C_2))$ *then Transform*$(C_2, S_2, Receive(Choicem(r', r, S_2)))$ $\forall r' \in SR(C_2)$;

Rule 18: The controlflow concept is transformed into a transition, for a participant role *r* in C_1. This transition connects the states S_1 and S_2 obtained from the transformation of C_1 and C_2 to the control states. When a guard is associated to the controlflow, the guard is associated to the generated transition in the derived model of the starting role.

If $r \in PR(C_1)$ *then Transform (Controlflow, Transition);*

Rule 19: The DecisionNode concept is transformed into a ChoiceState, for a participant role *r* in C_1. This state is connected to the composite states S_1 and S_2 obtained from the transformation of C_1 and C_2, by the transitions obtained by application of Rule 18.

If $r \in PR(C_1)$ *then Transform(DecisionNode, ChoiceState);*

4.6. While loop with Weak sequencing (local choice)

There is weak sequencing between the end of C_1 and the choice of executing C_1 again or terminating the loop with C_2 (Figure 8). The transformation of the collaboration concept for a starting role in the collaboration 2, a participant role and not a starting role in the collaboration 2, and a participant role in the collaboration 1 and not participating in the collaboration 2 requires triggering respectively the rules 15, 16 and 17. The transformation of the concepts Controlflow and DecisionNode for a participant role in the collaboration 1 triggers the rules 18 and 19 respectively.

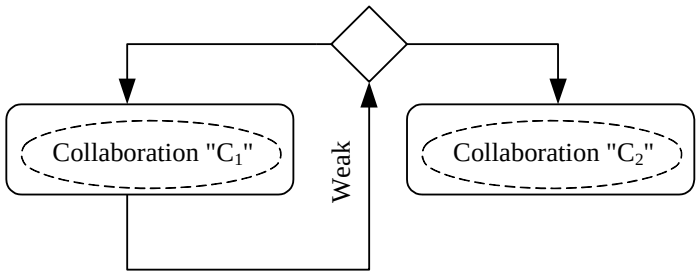

Figure 8. While loop with weak sequencing

Rule 20: The collaboration concept C_1 is transformed into the composite state S_1, for a participant role *r*. This state holds the actions performed by *r*.

If $r \in PR(C_1)$ *then Transform*(C_1, S_1);

4.7. Parallelism between two collaborations

In the case where the role *r* is involved only in one of the two collaborations (Figure 9), the control nodes and the control flow are not transformed but only the concerned collaboration.

Rule 21: The collaboration C_i is transformed into the composite state S_i, for a participant role *r* in C_i. This state holds the actions performed by *r*.

If $r \in PR(C_i)$ *then Transform*(C_i, S_i) $\forall i=1,2;$

Rule 22: The Controlflow concept is transformed into a transition, for a participant role *r* in both collaborations C_1 and C_2. This transition connects the states S_1 and S_2 obtained from the transformation of C_1 and C_2 to the control states.

If $r \in PR(C_1)$ *and* $r \in PR(C_2)$ *then Transform (Controlflow, Transition);*

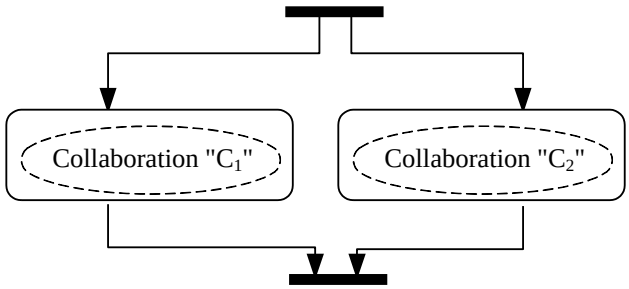

Figure 9. Parallelism between two collaborations

Rule 23: The ForkNode concept is transformed into a ForkState, for a participant role *r* in both collaborations C_1 and C_2. This state is connected to the states S_1 and S_2 obtained from the transformation of C_1 and C_2 by the transitions obtained by Rule 22.

If $r \in PR(C_1)$ *and* $r \in PR(C_2)$ *then Transform (ForkNode, ForkState);*

Rule 24: The JoinNode concept is transformed into a JoinState, for a participant role *r* in both collaborations C_1 and C_2. This state is connected to the states S_1 and S_2 obtained from the transformation of C_1 and C_2 by the transitions obtained by Rule 22.

If $r \in PR(C_1)$ *and* $r \in PR(C_2)$ *then Transform (JoinNode, JoinState);*

5. Case study

In this section, we present an application of telediagnosis in neuroscience. Conventionally, when a patient with a stroke is admitted into a hospital (HA), he will be examined by an emergency doctor. The emergency doctor contacts the EMS (Emergency Medical Services) via a regulating doctor to inform him of the emergency admission of a patient who presents symptoms to suspect a stroke (speech disorders, vision disorders, sensor-motor deficits, coordination disorders, *etc*…). Within the EMS information system, the regulating doctor creates a medical record based on an initial evaluation using the GCS (Glasgow Coma Score), the hemodynamic status, time of occurrence of signs, time of admission to emergencies, background, anamnesis data and clinical examination, *etc*.

While remaining in contact with the emergency doctor, the regulating doctor of the EMS calls the neurologist on duty at UHC (University Hospital Center) and initiates a conference call in order to establish the diagnosis. If appropriate, the patient is urgently transferred to the

University Hospital Center within an equipped ALS (Advanced Life support) or not-equipped ACA (Ambulance Care Assistance) ambulance depending on the patient health situation. The derivation process follows two steps.

The first step specifies the global requirements model describing the system global behavior. The system global behavior is modeled by an activity diagram whose core activities are collaborations and sub-collaborations (Figure 10). It is described by a collaboration consisting of a weak sequence between the *Clinical* sub-collaboration and the composite collaboration named *Clinical-Decision*. The *Clinical-Decision* collaboration is composed of a weak sequence between the *Para-clinical* sub-collaboration and the *Decision* collaboration. The *Decision* collaboration is itself composed of a weak sequence between the *Decision-Making* sub-collaboration and the *During-Transfers* collaboration. The last one consists of a choice composition between the collaborations: *Supported by HA* and *Transfer* which is also a choice composition. The collaboration *Transfer* defines a choice between the two sub-collaborations *Sending ALS* and *Sending ACA*.

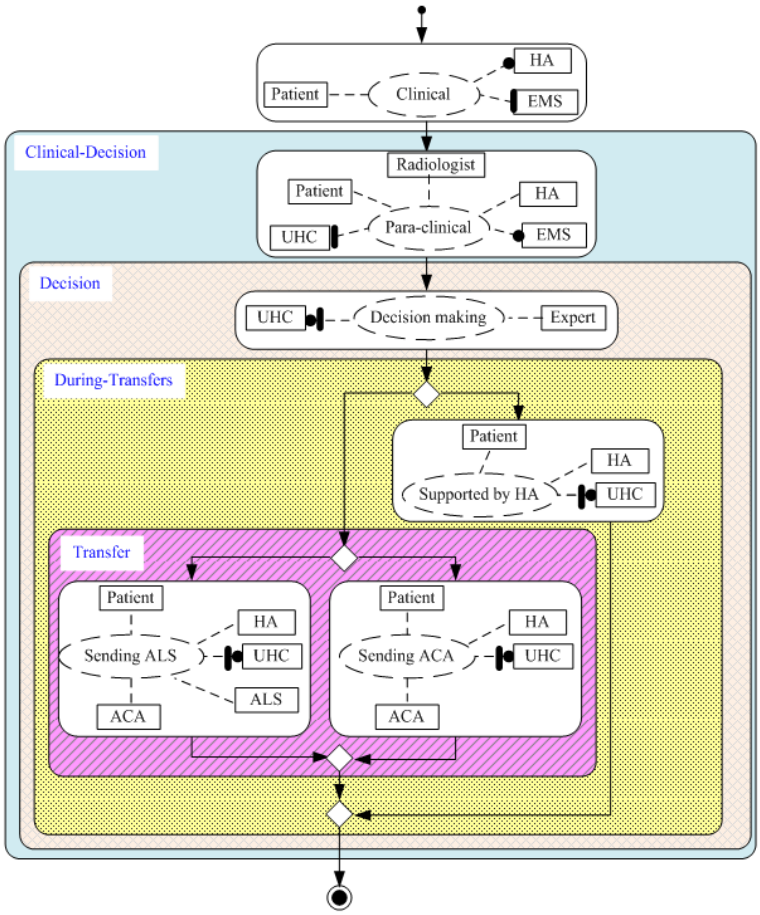

Figure 10. The system global requirements

The second step consists in applying the models transformation rules to the system global requirements model according to the derivation algorithm. These rules generate a state machine describing each role behavior. The derived state machine includes the messages required for ensuring the global coordination among the different system roles. We consider

in this example the transformation rules that allow deriving the behaviors of the equipped ambulance (ALS) and the neurologist (UHC).

The ALS behavior derivation can be summarized in the transformation of *During-Transfers* collaboration because the ALS does not participate in the *clinical*, *Para-clinical* and *Decision making* sub-collaborations. At the *During-Transfers*, the ALS is a participant role in *Transfer* collaboration but not in the *Supported by HA* sub-collaboration. Therefore the derivation process triggers the rules 7, 8, 9, 10 and 11. The rule 7 realizes the transformation of the *Transfer* collaboration. The rule 8 allows the ALS to receive the coordination message sent by the neurologist in case of choice of the *Supported by HA* sub-collaboration. The reception of this message triggers the continuation of the treatment at this sub-collaboration. Rules 9, 10 and 11 perform the transformation of the control flow and the control nodes. The transformation of the *Transfer* collaboration which is a choice composition is done in similar manner as the previous situation. As the ALS participates in the *Sending ALS* sub-collaboration and not in the *Sending ACA*, the derivation process triggers the rules 7, 8, 9, 10 and 11. These rules accomplish the transformation of the actions performed by the role as a participant at the *Sending ALS* sub-collaboration, allow the reception of the coordination message in case of choice of the *Sending ACA* sub-collaboration and perform the transformation of the control flow and the control nodes respectively. The derivation process generates a state machine which describes the ALS behavior (Figure 11).

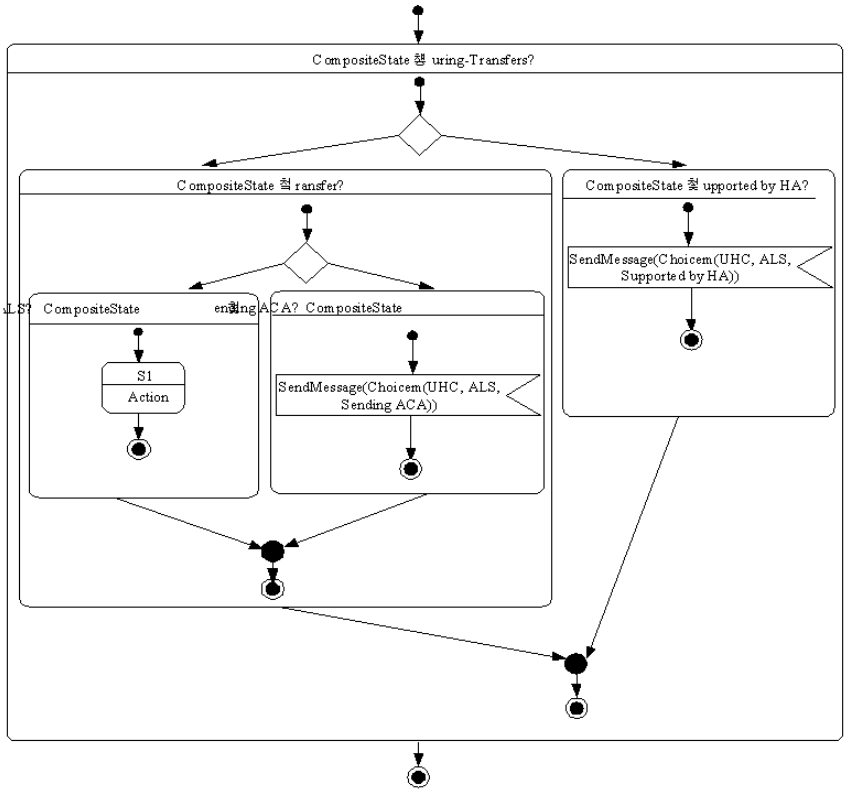

Figure 11. The derived model describing the ALS behavior

The neurologist (UHC) behavior derivation is obtained with the same manner as described in the ALS derivation. The (Figure 12) shows the derived UML state machine describing the UHC behavior.

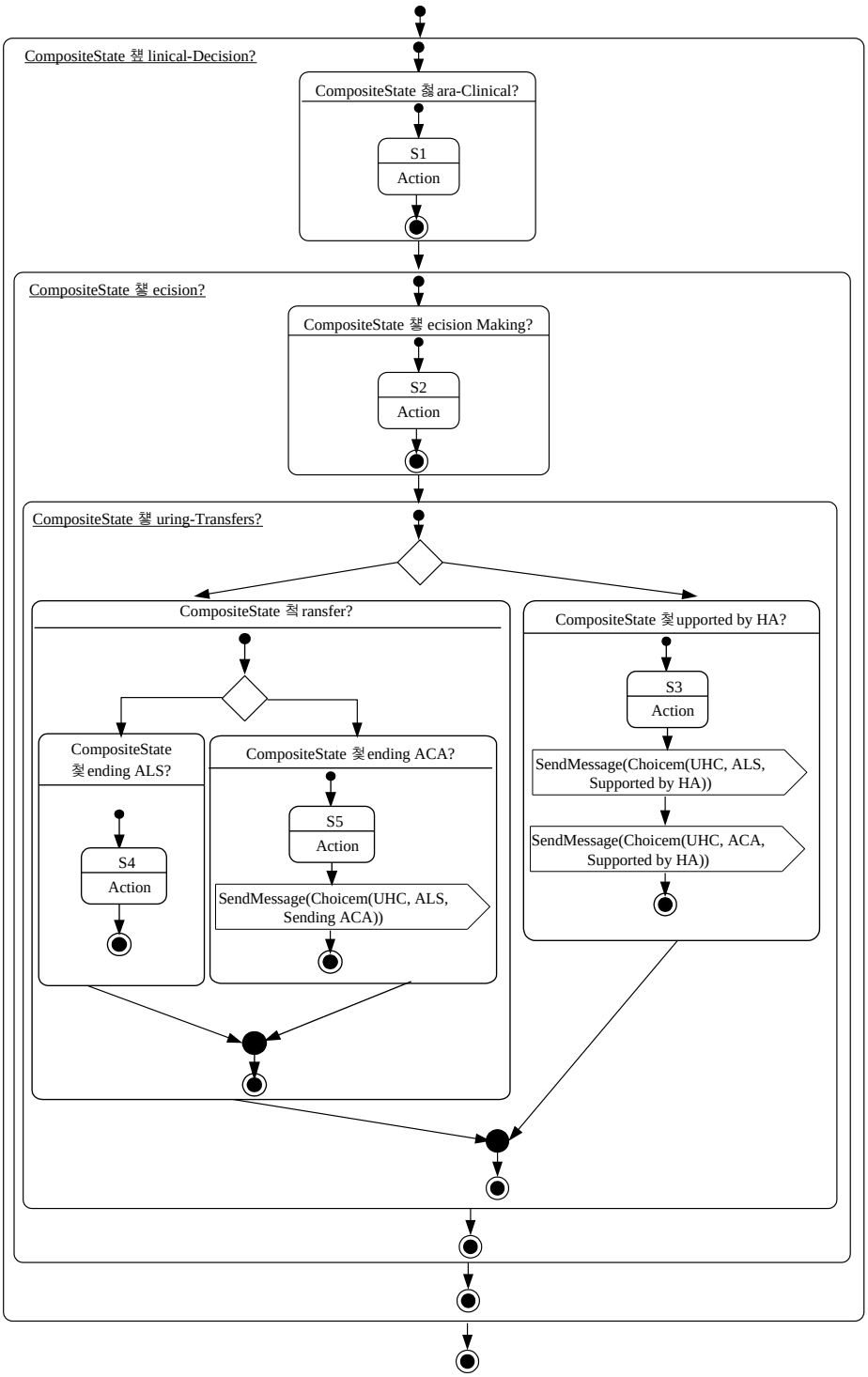

Figure 12. The derived model describing the UHC behavior

6. Implementation of the derivation process architecture

The meta-models and model transformations outlined in the previous sections have been developed using the facilities provided by the Eclipse platform. Eclipse platform is a free open source environment. It offers some widely used implementation of the OMG standard Meta Object Facility (MOF) [17], called Eclipse Modeling Framework (EMF) [18]. The architecture (Figure 13) shows the needed components to implement this development process. The architecture consists in three blocs:

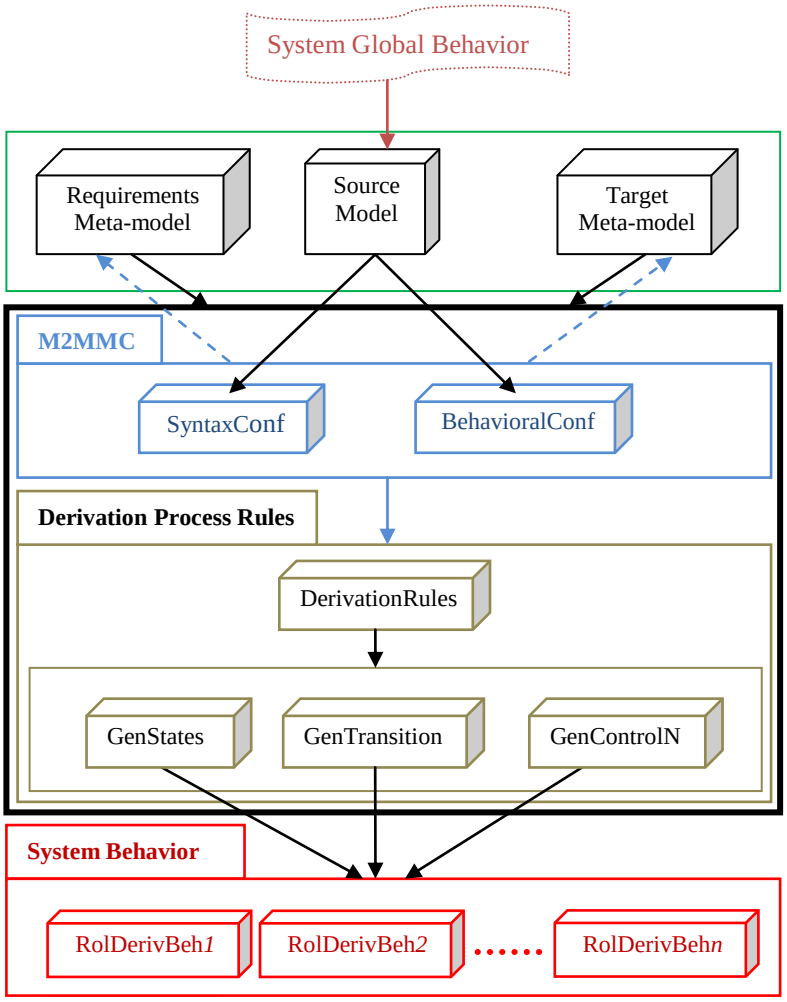

Figure 13. The derivation process architecture

1. The first bloc consists in representing:

   (i) The requirements meta-model, UML activity diagram extended with collaborations,

   (ii) The design meta-model, UML state machine meta-model, and

   (iii) The source model which represents a given system global requirements.

These meta-models and models are created using EMOF (Essential MOF). EMOF allows designers to create, manipulate and store both models and meta-models.

2. The second bloc is the main component of this architecture. It is responsible for the behavior derivation. It consists in: the *model to meta-model* conformity component, named *M2MMC* and the *Derivation process rules*. The *M2MMC* carries out the compliance of the source model with the requirements meta-model. This compliance is achieved by the EMOF. The compliance of the derived behavior of the components with the target meta-model is guaranteed by the *Derivation process rules*. The *Derivation process rules* performs the model to model transformation. It realizes the behavior derivation of the different components from the system global requirements model. This derivation is designed to generate the different concepts of a state machine for each system component.
   There exist several languages for model transformations. Among these languages, we mention the ATL language (ATLAS Transformation Language) [2] [19-20] and the QVT language (Query View Transformation) of the OMG [21-22]. Both languages exhibit a layered architecture and share some common characteristics as they initially shared the same set of requirements defined in QVT RFP [23]. ATL and QVT languages have a similar operational context [24-25]. The transformation rules of the derivation process are expressed in ATL language which provides the standard Eclipse solution for model to model transformations.

3. The third bloc consists in the derived UML state machines that reflect the local behavior of each component. These models are platform independent which could be used for an automatic generation of code for multiple platforms.

7. Conclusion

The work presented in this paper offers a model driven approach to derive the behavior of a distributed system components from its global requirements. The suggested approach presents a high level of abstraction using UML activity diagrams extended with collaborations. This allows designers to describe the system global behavior model. This approach allows representing concepts that could not be described using behavior expressions. The derived behavior of the system components are UML state machines which could be used for an automatic generation of code. Automatic model transformations from the requirements models to UML state machine models have been designed to govern the derivation process. They have been implemented using the ATL language “Atlas Transformation Language”. We have considered the transformation of the conditions (guards) embedded in the choice structures and the repetition structure. In addition, our transformation approach considers also all control nodes specified in the UML activity diagram such as *Merge Node* and *Join Node*. The derived state machine models include the messages required for realizing the collaboration and for ensuring the global coordination among the different system roles. An Emergency Medical collaborative application has been used to test and illustrate the development approach. We plan also to extend the derivation process by including the derivation of the detailed behavior of a sub-collaboration. Such behavior could be described

as a sequence diagram. The derivation of a sub-collaboration behavior will be achieved by transforming the sequence diagram into a state machine using a similar model transformations mechanism. Another future extension is to define a model to model transformation from the derived platform independent models to different platform models. This will allow us to define a model to text transformation which automatically generates the final code.

References

[1] D. Bertolini, L. Delpero, J. Mylopoulos, A. Nivikau, A. Orler, L. Penserini, A. Perini, A. Susi and B. Tomasi, "A Tropos model driven development environment", Proceedings of CAiSE 2006 Forum, **(2006)**.
[2] F. Jouault and I. Kurtev, "Transforming Models with ATL", Proceedings of MODELS 2005 Workshops, LNCS 3844, **(2006)**, pp. 128-138.
[3] J. Pavon, J. G. Sanz and R. Fuentes, "Model driven development of multi-agent system", LNCS, Springer-Verlag, vol. 4066, **(2006)**, pp. 284-298.
[4] A. Perini and A. Susi, "Automating model transformations in agent-oriented modeling", Proceedings of the 6th International Workshop AOSE, **(2005)**.
[5] S. Rougemaille, F. Migeon, C. Maurel and M. Gleizes, "Model driven engineering for designing adaptive multi-agent systems", Proceedings of the International Workshop on Engineering Societies in the Agents World (ESAW 2007), Springer-Verlag, **(2007)**.
[6] J. Sanz and J. Pavon, "Methodologies for developing multi-agent systems", Journal of Universal Computer Science, vol. 10, **(2004)**, pp. 359-374.
[7] V. Silva, B. Demaria and C. Lucena, "An MDA-based approach for developing multi-agent system", Proceedings of the 3rd Workshop on Software Evolution through Transformations (SeTra 2006) at the 3rd International Conference on Graph Transformation , vol. 3, **(2006)**, pp. 115-126.
[8] V. DeCastro, E. Marcos and J. Vara, "Applying CIM-to-PIM model transformations for the service-oriented development of information systems", Journal Information and Software Technology, vol. 53, **(2011)**, pp. 87-105.
[9] M. Karlos and M. Drosdova, "Analytical Method of CIM to PIM Transformation in Model Driven Architecture (MDA)", Journal of Information and Organizational Sciences, vol. 34, no. 1, **(2010)**, pp. 89-99.
[10] A. Rodríguez, E. Fernández –Medina and M. Piattini, "Towards CIM to PIM Transformation: From Secure Business Processes Defined in BPMN to Use-Cases", Proceedings of the 5th international conference on Business process management, LNCS, vol. 4714, **(2007)**, pp. 408-415.
[11] M. Lucena, J. Castro, C. Silva, F. Alencar and E. Santos, "STREAM: A Strategy for Transition between Requirements Models and Architectural Models", Proceedings of SAC 2011, ACM 978-1-4503-0113-8/11/03, TaiChung, Taiwan, **(2011)**.
[12] G. V. Bochmann, "Deriving component designs from global requirements", Proceedings of the International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES), Toulouse, France, **(2008)**.
[13] G. J. Holzmann, "The SPIN Model Checker: Primer and Reference Manual", Addison-Wesley Publishing Company, ISBN 0-321-22862-6, **(2004)**.
[14] Object Management Group, "UML 2.0 Superstructure Specification", OMG Adopted Specification, at http://www.omg.org , **(2007)**.
[15] H. Castejon, G. V. Bochmann and R. Brack, "Realizability of Collaboration-based Service Specifications", Proceedings of Asia-Pacific Software Engineering Conference (APSEC), Nagoya, Japan, **(2007)** November.
[16] H. Castejon, G. V. Bochmann and R. Brack, "Using Collaborations in the Development of Distributed Services", Department of Telematics 2008, 37 s. AVANTEL Technical Report, **(2008)**.
[17] Object Management Group, "Meta-Object Facility (MOF) 2.0 Core Specification", http://www.omg.org/docs/ptc/04-10-15.pdf, **(2004)**.
[18] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick and T. J. Grose, "Eclipse Modeling Framework", Addison-Wesley Professional, **(2003)**.
[19] F. Jouault, F. Allilaire, J. Bézivin and I. Kurtev, "ATL: a model transformation tool", Journal Science of computer programming, vol. 72, no. 1-2, **(2008)**, pp. 31-39.
[20] "The Atlas Transformation Language (ATL) project", at http://www.eclipse.org/m2m/atl/.
[21] Object Management Group, "MOF Query/ View/ Transformations (QVT) Adopted specification", http://www.omg.org, **(2005)**.
[22] Object Management Group, "MOF Query/ View/ Transformations (QVT) Adopted specification", http://www.omg.org, **(2008)**.

[23] Object Management Group, "MOF 2.0 Query/ Views/ Transformations RFP", OMG Document ad/2002-04-10, 2002, at http://www.omg.org/cgi-bin/doc?ad/2002-04-10, **(2002)**.
[24] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev and P. Valduriez, "ATL: a QVT-like Transformation Language", Proceedings of Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, ACM Press, **(2006)**, pp. 719-720.
[25] F. Jouault and I. Kurtev, "On the architectural alignment of ATL and QVT", Proceedings of the 2006 ACM symposium on Applied computing, ACM Press, **(2006)**, pp. 1188-1195.

Authors

Ahmed Harbouche has been an Assistant professor in Computer Science. He obtained his Magister in 1993 at Houari Boumedienne University (Algiers, Algeria) in the area of Artificial intelligence. He is member of the research team "models engineering" of the department of computer science at Hassiba ben Bouali University (Chlef Algeria). His research interests pertain artificial intelligence, multi-agents systems and Software Engineering.

Mohammed Erradi has been a Professor in Computer Science since 1986. He has been leading the distributed computing and networking research group since 1994 at ENSIAS of Mohammed V-Souissi University (Rabat Morocco). His recent main research interests include Communication Software Engineering, Distributed Collaborative Applications, Security Policies, Reflection and Meta-level Architectures. He obtained his Ph.D. in 1993 at University of Montreal in the area of Communicating Software Engineering under the supervision Professor Gregor Von Bochmann. He is leading, at the present time, TENEMO project (A Collaborative Environment for Neuroscience Tele-diagnosis over a Mobile Platform) funded by a French-Moroccan joint research program (2008-2011). He also is currently the Principal Investigator of a number of research projects grants. Professor Erradi has published more than 60 papers in international conferences and journals. He has organized and chaired four international scientific events and has been a member of the program committee in multiple international conferences.

Aicha Mokhtari is currently a Professor in USTHB Alger's University, Computer Science Department. Her research focuses on reasoning about knowledge and uncertainty, and its applications to security, web semantic, recommender system and data bases. Her work lies at the boundary of a number of fields. I usually teach data bases in an undergraduate course and knowledge representation and reasoning in a postgraduate course.