From Graphical Model in UML Activity Diagrams to Formal Specification in Event B for Workflow Applications Modeling

Ahlem Ben Younes¹, and Leila Jemni Ben Ayed²

¹Research Unit of Technologies of Information and Communication (UTIC)- ESSTT-Tunisia
Ahlem.benyounes@fst.rnu.tn

²Research Unit of Technologies of Information and Communication (UTIC)- ESSTT-Tunisia
Faculty of Science of Tunis
Leila.jemni@fsegt.rnu.tn

Abstract — The lack of a precise semantics for UML AD makes the reasoning on models constructed using such diagrams infeasible. However, such diagrams are widely used in domains that require a certain degree of confidence. Due to economical interests, the business domain is one of these. To enhance confidence level of UML AD, this paper provides a formal definition of their syntax and semantics. The main interest of our approach is that we chose UML AD, which are recognized to be more tractable by engineers. We outline the translation of UML AD into Event B in order to verify functional properties of workflow models (such as deadlock-inexistence, liveness, fairness) automatically, using the B powerful support tools like B4free. We propose a solution to specify time in Event B, and by an example of workflow application, we illustrate the proposed technique.

Index Terms—Specification, Formal verification, Validation, UML, Event B, workflow application

I. INTRODUCTION

The work presented in this paper is part of our works [2][13] that aims at providing specification and verification technique for workflow applications. A workflow is an operational business/scientific process. In order to represent workflows in an intuitive and practical way with a standard language, we have chosen UML AD. The modeling process is not addressed in this paper; more details about it can be found in [2]. However, the fact that UML lacks a precise semantics is a serious drawback of UML-based techniques. Also, UML AD is not adapted to the verification of workflow applications. In this paper, our goal is to provide a specification and verification technique for workflow applications using UML AD endowed with timed characteristics and synchronisation aspects. In fact, in the business domain, the emphasis is made on the response time properties that guarantee a quality of the service. The proposed approach gives readable models and an appropriate formal method which allows verification of required properties (no deadlock, liveness, fairness) to prove the correctness of the workflow specification. In this context, several solutions have been proposed. Some of them use model checking for the verification. Van der Aalst [10] proposed a technique which uses Petri nets for the verification of the correctness of workflow applications using a compositional verification approach. Karamanolis et al [11] use process algebra for the verification of workflow properties. We have chosen to use the event B method and its associate refinement process and tools for the formal verification of workflow applications. The verification is based on a proof technique and therefore it does not suffer from the state number explosion occurring in classical model checking as in the cases of works in [10] and [11], which propose approaches for the verification of the correctness of the workflow specification.

In our previous work [2], we have proposed an approach which combines the use of UML AD and Event B for the specification and the verification of workflow applications. In this paper, we extend our work presented in [2][13] by adding new translation rules for the UML AD endowed with timed characteristics and synchronisation aspects into Event B. Also, as in general the business domain depends on time consideration in their functionalities, we propose in this paper a solution to specify time in the event B method and derivation of temporal expressions in UML AD (timeout) into Event B. The result of the translation allows verification of the workflow termination. These translation rules give not only a syntactical translation, but also give a formal semantics using the Event B method semantics for the activity diagrams. In this context, there have been efforts for defining semantics for activity diagram in the works of Eshuis [7][12]. However, these works not consider the hierarchical decomposition of activities in UML AD, and suffer from the state number explosion. Moreover, in Eshuis [12][7] approach, no details are given about how time is defined. This paper is structured as follows, Section 2 describes the translation of hierarchical decomposition of UML AD into a hierarchy of Event B models. It presents derivation rules of time in UML AD into Event B notation. By an example we illustrate our contribution. Finally, a summary of our work concludes the paper.

II. THE PROPOSED APPROACH
As shown in Figure 1, the proposed approach consists mainly of four steps. In the first step, the workflow is modeled graphically with UML AD. In the second step, the resulting graphical readable model is translated into Event B in incremental development with successive refinements. This refined model is enriched by relevant properties (no deadlock, no livelock, strong fairness, etc) (Step3) which will be proved using the B4free tool [6] (step4). The verification of these properties ensures the correctness and the validation of the described workflow. In [2][13], we have proposed translation rules for the concepts of UML AD (activity, Sequence of activities, correctness and the validation of the described workflow.

Also, we have proposed, in [8], translation rules of event, send/receive event action concepts in UML AD into Event B. Due to space limitation we will not present all the proposed rules but just our proposed solution to the timeout concept in UML AD into Event B which will be illustrated over the ATM Login application example.

**Step 1: Specification with UML AD**

Initially, we describe this workflow application using UML AD by employing a refinement technique [2]. The resulting model is composed of three decomposition levels (See Figure 2). Each activity has only one id (ATM, Card_Details, Eject_Card, User_PIN, Check_PIN, Get_out, Select_Transition, Get_Pin, Valid_PIN, Abounding, and End_Time).

**Step 2: Translation into Event B.** Our proposed translation process uses the refinement process of Event B to encode the hierarchical decomposition in the UML AD: to each decomposition level in UML AD, is associated an Event B refinement.

The representation of the time in Event B

In UML, the time is incremented by means of an internal clock. The time is not a primitive of Event B. Specifying time in B, needs to add a clock in alternation with the system. We propose to model the action of this clock in Event B by the definition of a B event Tick. The Tick’s action consists of advancing the time represented by a B variable Time of integer type that we define in the build B system. The B event Tick maintains the control and allows time advance. In this way, we avoid the Livelock problem in the construction of a resulting B system. The timeout expressed in B, will impose alternation between the clock, the control system and the detection system. We use the variable hand [8] and the control is given alternatively to the clock when hand=2 and to the system in the other cases.

In UML AD, temporal event is specified with the after keyword. The after(n) event expression, where n is a positive integer, means that n time units after the source of the edge was entered a timeout is generated. The following figure shows an example.

**Translation Rule of the event Timeout**

The formulation of the timeout in Event B is based on some derivation rules that we already introduced. We propose to drift the timeout by:

- the definition of a variable integer Timeout which represents the time of the next generation of the event timeout;
- the definition of the event Evt_Init for initializing the variable Timeout with the value of the current time and the duration n ( Example n =2 time unit).

the definition of the B event Evt_Wait.

![Figure 1. Derivation process from UML AD to Event B](image-url)
Initially, hand = 0. The event Evt_Init sets this variable to 1, and initializes the variable timeout. This passes the control to Evt_wait or Evt_Act01. If (time < timeout) then the variable hand passes to 1 (event Evt_Wait) to increment the time (event tick) but if (time >= timeout) this (Evt_Act01) is allows the following activity Act01 to be execute.

By the application of the translation process and using the translation rules [2][13], the initial UML AD model of the ATM Login application, in figure 2, is translated into B event in a set of property preserving refinements. Three refinement steps which correspond to each level of decomposition in the UML AD model (Figure 2) are necessary. Following figure 2, the activities Card_Details, Eject_Card, Get_out, Select_transaction, Get_PIN, Valid_PIN, Abounding, and End_Time correspond to basic process/tasks of ATM Login application. We generate for the abstract level LEVEL0 an Event B model ATMcard, for the first decomposition level LEVEL1 an Event B refinement Ref1_ATMcard, for the second decomposition level LEVEL2 an Event B refinement Ref2_ATMcard, and for the third decomposition level LEVEL3 an Event B refinement Ref3_ATMcard. In following, we focus on the translation of event timeout.

In the refinement Ref3_ATMcard, the event Tick maintains the control and allows time advance. In this way, we avoid the Livelock problem in the construction of a resulting B system. By the application of the translation rule for the time, we use the variable hand and the control is given alternatively to the clock when hand = 2, to the system when hand = 0, and to the detect system when hand = 1. The variable hand describes the events interleave and prevent that an event is fired infinitely (an event will be infinitely crossed in detriment of others).

**Step 3: Fill up the system with properties**

In this step, we enrich the models with invariants/Assertions describing required properties. The ASSERTIONS clause contains liveness properties expressing that there is no deadlock. This property is ensured by asserting that the disjunction of all the abstract events guards implies the disjunction of all the concrete events guards. This guaranties that the new events can be fired (no deadlock). In each new refinement, we add this property: for example in the Ref2_ATMcard:

The INVARIANT clause allows to express the safety properties (called safety invariant) and the typing information (Typing invariant). Each refined model is enriched by relevant properties (safety, liveness, etc) which will be proved using the B4free tool. These properties shall remain true in the whole model and in further refinements: It is not needed to re-prove again verified properties in the refined model while the model complexity increases. It is the advantage of using B4free tool:

- The safety property that the system ejects the card reader only if the card details are false: this property is added in the resulting refined model Ref1_ATMcard (associated to the LEVEL1) in the clause INVARIANT as follows:

**Refinement Ref1_ATMcard**

- **INVARIANT** /* Safety properties*/
  (card_reader_eject = true => card_detail = false)

- The temporal property T1 (the system should not be continuously open for more than 10 seconds without the even PIN present) can be proved by adding the safety invariant, in the resulting refined model Ref3_ATMcard (associated to the LEVEL3 in UML AD model), which expresses
that if the system is in the node Abounding (pin_state = 1), then necessarily the deadline has arrived:

```plaintext
REFINEMENT Ref3_ATMcard ...
REFINE Ref2_ATMcard
INVARIANT /*Temporal properties*/
(pin_state = 1) ⇒ (time >= timeout) /* T1*/
```

The strong fairness (no livelock) properties are expressed by the events interleave by using the variable hand that solve livelock problem. The guards of these events define their firing order and how these events interleave.

**Step 4: Validation/ Verification**

A ATM subactivity is described by an initial state and a final state. It is refined into a sequence of basic events which lead from the initial state to the final one. The refinement preserves all the properties of the initial activity. This process is repeated until basic events are reached. When the basic events are reached by the refinement, the validation process is completed. The validation of the ATM activity allows to express reachability properties. For example, if a ATM activity is validated, then the objective of the application which consists of executing activity Card_Details and activity User_PIN or Eject_Card is realisable. In addition, for example this allows to express that while the event PIN is not detected in 10 s, after the execution of the activity Get_PIN, the activity Valid_Pin can not be executed.

The following table1 illustrates the obtained results on our case study ATM login.

<table>
<thead>
<tr>
<th>Model</th>
<th>nObv</th>
<th>nOp</th>
<th>nAuto</th>
<th>nInt</th>
<th>%Pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Ref1_ATMcard</td>
<td>32</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Ref2_ATMcard</td>
<td>84</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Ref3_ATMcard</td>
<td>255</td>
<td>25</td>
<td>25</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td><strong>TOLAL</strong></td>
<td><strong>371</strong></td>
<td><strong>39</strong></td>
<td><strong>39</strong></td>
<td><strong>0</strong></td>
<td><strong>100%</strong></td>
</tr>
</tbody>
</table>

Table 1. Summary of proofs, all Proof Obligations generated have been proved (P= 100%)

The resulting Event B specification has been proved totally. All proof obligations (100%) are proved automatically, and then the initial UML AD model of our ATM workflow application is validated.

**III. Conclusion**

In this paper, we have presented a formal syntax and semantic for UML AD endowed with time aspect and synchronisation (send/receive event). A systematic way for translating this semantics into Event B notation is also provided. Such a translation is not an end in itself, it is a basis for a formal and automatic verification of wide range of constrains including lives, no_deadlock, and safety properties, with the B support tool B4free, that ensure a confidence level for workflows applications such as business process. Currently, we are working on the implementation of this approach. Another thing needed to be mentioned is that we just formalize the subset of UML activity diagrams. For instance, object flows do not be included in our model. However, our approach is also suitable for formalizing it.

**References**