From Business Processes to Integration Testing

A public transport case study

Stéphane Debricon [*] — **Fabrice Bouquet**[*] — **Bruno Legeard**[**]

[*] *LIFC*
Université de Franche-Comté
16 Route de Gray
25030 Besancon Cedex
{stephane.debricon,fabrice.bouquet}@lifc.univ-fcomte.fr

[**] *Smartesting*
TEMIS Innovation
18 Rue Alain Savary
25000 Besancon
legeard@smartesting.com

RÉSUMÉ. *Cet article se place dans le domaine de la génération de tests à partir de modèle (MBT). Celle-ci vise à fournir des campagnes de tests pour assurer la robustesse, l'adéquation aux besoins ou la sécurité d'une application. Dans la pratique, l'équipe en charge de la validation produit un modèle pour chacun des blocs fonctionnels présents dans le système à tester. Ce modèle est alors une source d'informations à prendre en compte pour produire de nouvelles campagnes de tests sur le système complet.*

Pour cela, nous ajoutons un nouveau niveau d'abstraction permettant la représentation de comportements métiers en utilisant les blocs fonctionnels déjà modélisés. En les associant et en ajoutant les objets échangés par ceux-ci, nous pouvons décrire des processus métiers qui n'étaient pas encore capturés aux niveaux des blocs.
Dans ce travail, nous proposons une approche méthodologique qui s'intègre au processus unifié (UP) et la gestion des conflits induits par la recombinaison des blocs. Nous illustrons notre proposition en l'appliquant à un cas industriel: un système de vente de titre de transport en commun.

ABSTRACT. *This paper is a contribution to the Model-Based Testing (MBT) field. MBT aims at producing tests suites that will be used to check for security, robustness or correct software*

adequacy with requirements expressed by customers. In practice a validation team produces a model for each functional block involved in a system under test. Thereby, the model gathers all kind of information that can be used to produce new tests suites for the complete system.

Therefore, we introduce a new abstraction level, allowing the description of new business behaviours using functional blocks already modeled. Those blocks are combined and augmented with objects exchange, producing business processes not captured at the block level. In this work, first we introduce a methodological approach fully integrated in the Unified Process (UP) and then a conflicts resolution due to block recombination. We illustrate our proposal with an industrial case study: a transit passes selling system.

MOTS-CLÉS : *Test à partir de modèles, Processus métiers, Test d'intégration, BPMN, UML*

KEYWORDS: *Model-Based Testing, Business Process, Integration Testing, BPMN, UML*

1. Introduction

Model-Based Testing (MBT) (Utting *et al.*, 2006b) aims at producing tests suites to check for security, robustness or correct software adequacy with requirements expressed by customers. This technology based on UML was used in our previous research to ensure functional coverage of software development (Utting *et al.*, 2006a). It was integrated in a development life cycle called Unified Process (UP) (Arlow *et al.*, 2003) to help its adoption and spreading.

In consideration of a practical application of this method, a development and validation team would produce a model containing functional components representing the system under test. Thereby, the model gathers all kind of information that can be used to reach a new step in tests generation. It contains behaviors, data and tests allowing non-regression testing. We propose to combine all those information to generate new tests suites. Those tests are based on business processes defined from model's elements. As a consequence, we introduce a new abstraction level, allowing the description of new behaviors not yet represented in the model.

Business processes can be seen as activities (or actions) sequences, that might not be described in initial requirements, but are known by experts to be interesting behaviors. Processes can be written by analysts from the validation team (as described in the method introduced in (Bouquet *et al.*, 2006)) or by domain experts. A business process is viewed as an integration test of model's components and is fully integrated in the testing process. A process is described by combining components already modeled, augmented with data information spreading from an activity to an other, to create an integration test. Our goal is not to automatically generate a business process, but rather to give the opportunity for an expert to model a specific and interesting behavior. However, a test model should be available for every components used in the business process, and we believe that this element's composition will produce tests suites for integration testing.

Our intention is to guide a business specialist or an analyst to describe business processes, which cannot be found in analysis model and to generate tests suites from them. Those tests complement tests already produced by automatic tests generation from behavioral models. Modelers can describe a business process or an integration test. A business process can be seen as an integration test since it models a sequence of behavior along with data flow.

The rest of this paper is organized as follow : Section 2 presents the methodology for modeling a business process from existing elements in the model. In this section this approach is applied on the case study. Section 3 details the production of business processes tests suites from the model. The section presents the initial state derivation and how to deal with data flows and control nodes of the business process activity diagram. Section 4 gives an overview on the tests generation technique and the number of tests produced for a simple example.

2. Business process modeling methodology

In this section we describe how to handle a business process and information required for its modeling in an UP analysis phase.

2.1. *Approach*

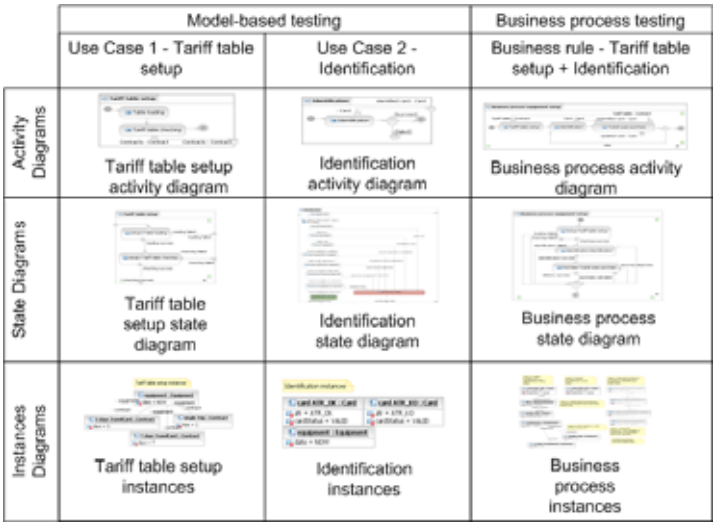


Figure 1. *Model elements reuse*

Usually analysis phase is split in the four following steps (Arlow *et al.*, 2003) : i) use cases modeling, ii) classes and objects detection, iii) relationships between classes and iv) objects definition and use cases realization. We propose to introduce business process in the model available at the end of the last step, linking use case realization with each other. Model also includes data flowing between use cases. From now on, we focus on use case realization. Each use case identified by previous steps of analysis phase are refined with a sequence diagram describing nominal behavior. This sequence diagram should come along with an activity diagram describing use case general behavior. The diagram contains every actions required for use case realization, it also underlines objects flowing from an action to an other. Furthermore, actions introduced in activity diagram, are refined into state machines. Those machines contain use case behaviors expressed by means of states and transitions augmented with OCL code. By the end of this phase, the model should contain activity diagrams for each use case previously identified. At this point our approach introduce business processes modeling. Business activity diagrams should link use cases with each other, through the reuse of their respective activities. Each activity diagram describing a business process is made of references to activities (use cases) of the model. Furthermore, the model will bring to light all data flowing between activities.

Activity diagrams will be used to represent the control flow of business processes and data flows between activities defining them. Activities themselves are described in terms of state machines which can be used for automatic tests generation as presented in (Bouquet *et al*., 2007). It remains to define an initial state based on data flows expressed in the business process activity diagram. The new business process is created from elements already defined in the model produced for model-based testing. The approach is summarized in Figure 1.

2.2. *Practical application : Transit passes selling software*

The approach described in the previous sub-section is applied on a transit passes selling machine real-life example. This example is the result of the project VALMI (validation of embedded micro-systems) which was carried out by Parkeon (http ://www.parkeon.com/), ERG Transit System (http ://www.erggroup.com/), Smartesting (http ://www.smartesting.com/) and LIFC (http ://lifc.univ-fcomte.fr/). Considering an urban transit system, a customer can buy a transit pass from a vending machine. Transit passes sold by the machine are stored in a contact-less smart card. Delivery steps are described in a standard document called Intercode II (French norm XP P 99-405 / European norm ENV1545), that contains information and processes to be used by urban project stakeholders. Delivery steps are as follow : contact-less smart card detection and identification and stored diagnostics handling. Diagnostics are special events created during the card life-cycle (for example : card invalidation due to dysfunction or e-selling from the back-office). Then the equipment starts the delivery activity that includes the choice of a transit pass, a quantity and payment. The complete model was produced using Rational Software Architect by industrial partners with the help of the LIFC.

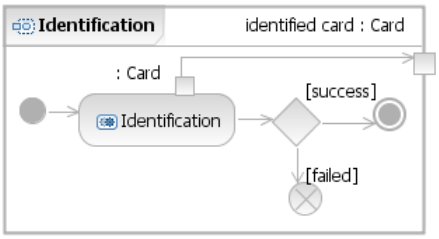
At first use cases of the selling application were identified. In this paper, we only consider two use cases : transit pass purchase and tariff table setup.

2.2.1. *Activity diagrams*

Two use cases of the application are selected and refined in this section. Activities are illustrated in Figure 2 and 3. We use two notions of the activity diagram ie the control flow to describe the behavior of the activity. In addition, we use the concept of data flow expressing the sharing of information from one activity to another.

Figure 2 shows an activity made of an action called *Identification*. This activity also presents the smart card processed as an output parameter, this data flow can be used to connect the activity to another one. There are two possible final control flows : *failed* or *success*. Only one of those can be connected to an other activity, i.e. the activity final node (success), the flow final node (failed) cannot be connected to any activity.

Activity presented in Figure 3 contains two actions required to complete tariff table setup. At first the tariff table should be loaded into the equipment, then it should be



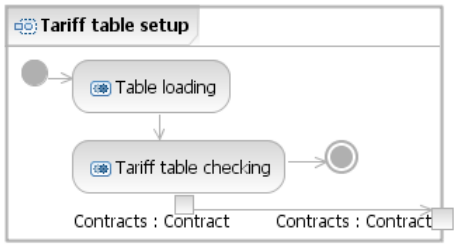


Figure 2. *"Card identification" activity*

Figure 3. *"Tariff table setup" activity*

verified by an technical operator. Once this checking is complete, the tariff table will be available for any other activity as a set of *Contract*.

The third activity (not illustrated) is the transit pass delivery, it is composed of two actions : *Diagnostics handling* (no details given in this paper) and *Delivery*. The activity has three parameters : two input parameters i.e. the smart card (detected by the equipment) and the tariff table (available in the machine). The smart card updated with the new transit pass purchased is an output parameter. The smart card is an activity's object which state may be changed by activity's actions. This activity contains an initial node and two final nodes, nominal case is represented by the control flow ending with the activity final flow. There is also an error case represented by a flow final node, stopping the activity.

2.2.2. *State machines*

Behaviors modeled in activity diagrams are described more precisely with state machines.

Figure 4 shows a state machine describing every steps required to setup the equipment, that is tariff table loading and checking. The first step is represented in Figure 5. This state diagram provides complementary information to activity diagram in Figure 3. It shows entry and exit pseudo-states to be used to model a business process. Thereby there are three exit pseudo-states : *loading failed*, *checking failed* and *checking success*.

2.2.3. *Business processes*

We have enough information in the model to define a business process. We focus on a process managing an equipment setup before its use. Figure 6 shows a business process activity diagram that gather activities previously introduced, augmented with data flows between activities.

A business process activity diagram contains a control flow describing the way activities are called. The sequence will be as follow : flow starts by business process initial node, then activates the first activity initial node. At the end of the current ac-

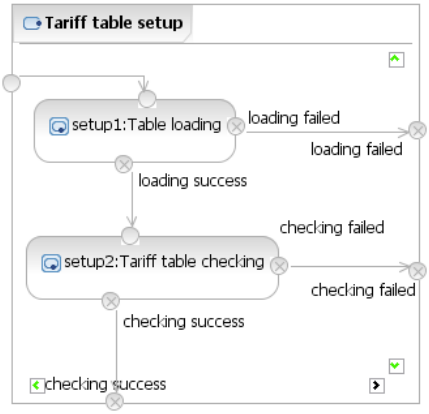


Figure 4. *"Tariff table setup" state machine*

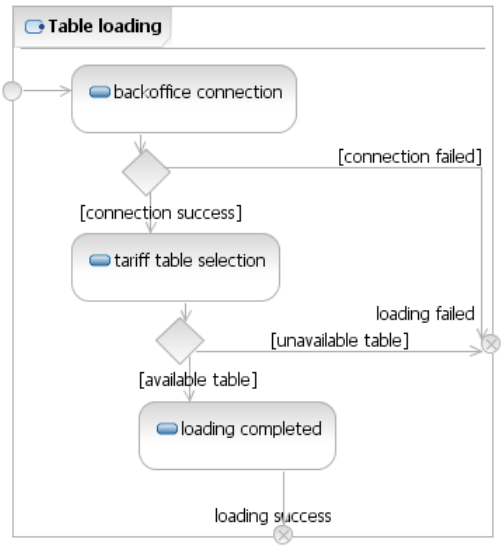


Figure 5. *"Tariff table loading" state machine*

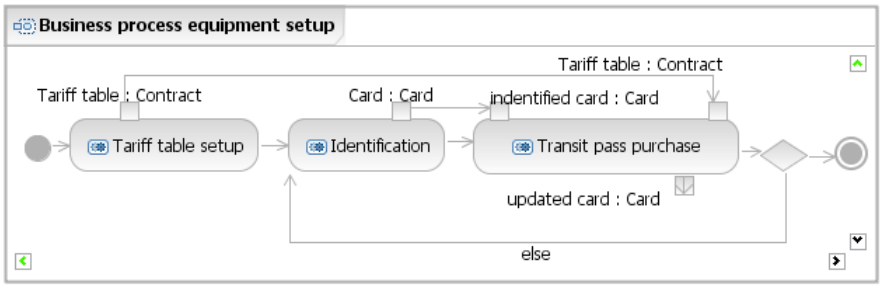


Figure 6. *Business process "Equipment startup and use" activity*

tivity (through activity final node and not any flow final nodes) the flow activates the next activity and so on. On Figure 6, process starts with activity called *Tariff table setup* then *Identification* followed by *Transit pass purchase*. The process goes back to the *Identification* activity waiting for a new card. Scenario ends at equipment shutdown.

The activity diagram also contains data flows between activities. Thereby *Tariff table setup* activity provides tariff table to *Transit pass purchase* activity. *Identification* activity describes smart card detection and recognition of a card used by *Transit pass purchase* activity. This card is not used in any other business process actions, as it is

not relevant for any situation described by the process and there is no need to spread this information.

We introduced a method based on the presence in the model of all necessary elements to create a business process (bottom-up method : from detailed elements to business process). It is also possible to produce all elements from the business process i.e. to describe the process then to detail step by step all behaviours as seen in previous section (down method : from business process to detailed element). A test generation is possible as soon as machines and initial states are produced. We recommend to reuse all knowledge included in the model but both methods (bottom-up or down) can be used to create business processes. A good practice is to mix both of them, that is to use pre-existing elements and to describe missing behaviours.

Providing data exchanged between activities is a key concept for tests generation. We will address in the next section key problems of exchange between activities : data sharing(exchange and initial states), control nodes and tests generation.

3. Tests suites generation for integration testing

In previous section, we have seen how to model a business process using activity diagrams augmented with data exchanged between activities. Scenarios are extracted from the diagram, to test correct integration of activities within the application.

Using the business process activity diagram, we produce a new state machine made of machines composition. The resulting machine is obtained by connecting several sub-machines already defined in the analysis model. Machines entry and exit points should be connected to create a new machine that will model system behavior within a business process. This splitting phase underlines and links several sub-machines connection points, particularly in the case of a machine with several exit points. Data exchanged by activities are not represented within resulting state machines but are only modeled in activity diagram as seen in the previous section. Thus each activity designed for MBT should come with a state machine, an initial state made of instances and a common class diagram.

We are not trying to introduce any automatic composition of state machines, the composition process should be dedicated to an analyst. His task should be to use any information available in the model to create a new diagram guided by business process. Figure 7 shows sub-machine *Tariff table setup* described in Figure 4, with two exit points connected to sub-machines : *Identification* and *Transit pass purchase* (no details of those machines given here).

3.1. *Deriving business process initial state*

The state machine introduced in previous section describes expected functional behavior of the system when activating a business process. The machine is made of

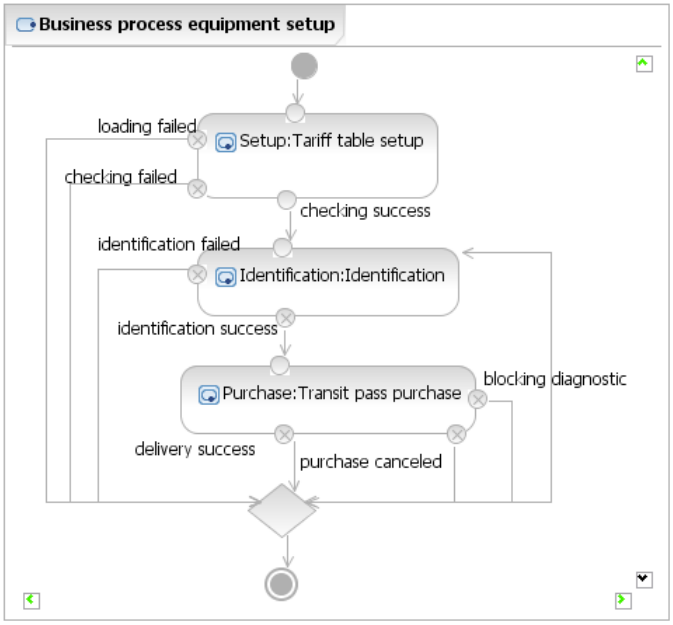


Figure 7. *Business process "Equipment startup and use" state machine*

sub-machines for which tests suites may be available. Those suites should have been independently validated. They come with an initial state made of instances which evolution, through the computation of transition post-conditions, define the expected verdict. Once we have connected our sub-machines, a new initial state must be specified using every sub-machines initial state. Furthermore, information available on business process activity diagram allow us to identify objects we want to retain from an activity to an other, so from a machine to an other.

Choosing values for business process initial state : Every sub-machine composing a business process has its own initial state (Figure 8, 9, 10). A new initial state made of states combination has to be created for the process state machine, we examine every instance diagram to identify instances and slot values to be retained in resulting initial state. To do so, instances must keep the same name (identifier) in the model. That is to say, an instance describing a specific contract, for instance, should have only one identifier. There are three kinds of instances in the model : system under test class instances, spread classes instances (used in business process activity diagram) and all other classes.

System under test class instances are shared by all initial states of the model. We must compare *slot values* in all instance diagrams to define which value to retain for the new initial state :

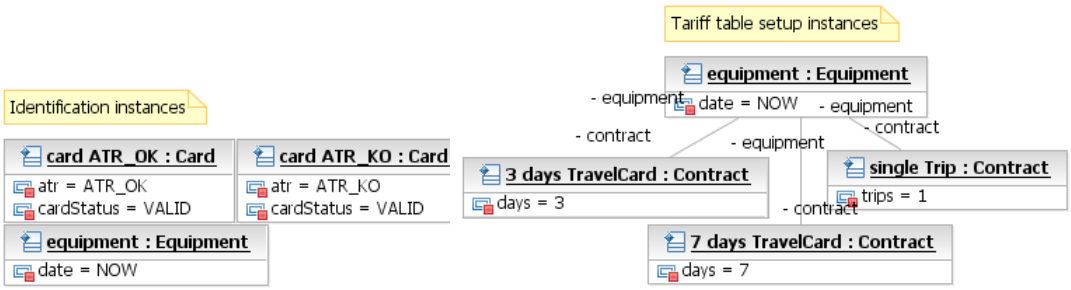


Figure 8. *"Identification" initial state diagram*

Figure 9. *"Tariff table setup" initial state diagram*

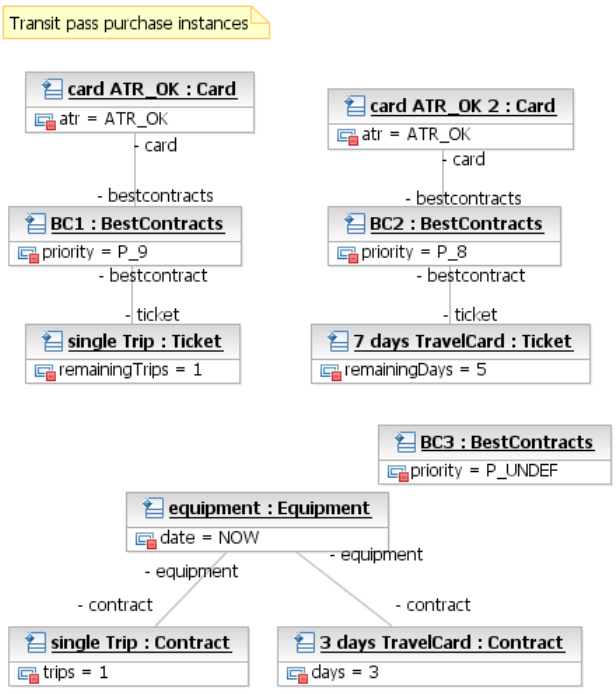


Figure 10. *"Transit pass purchase" initial state diagram*

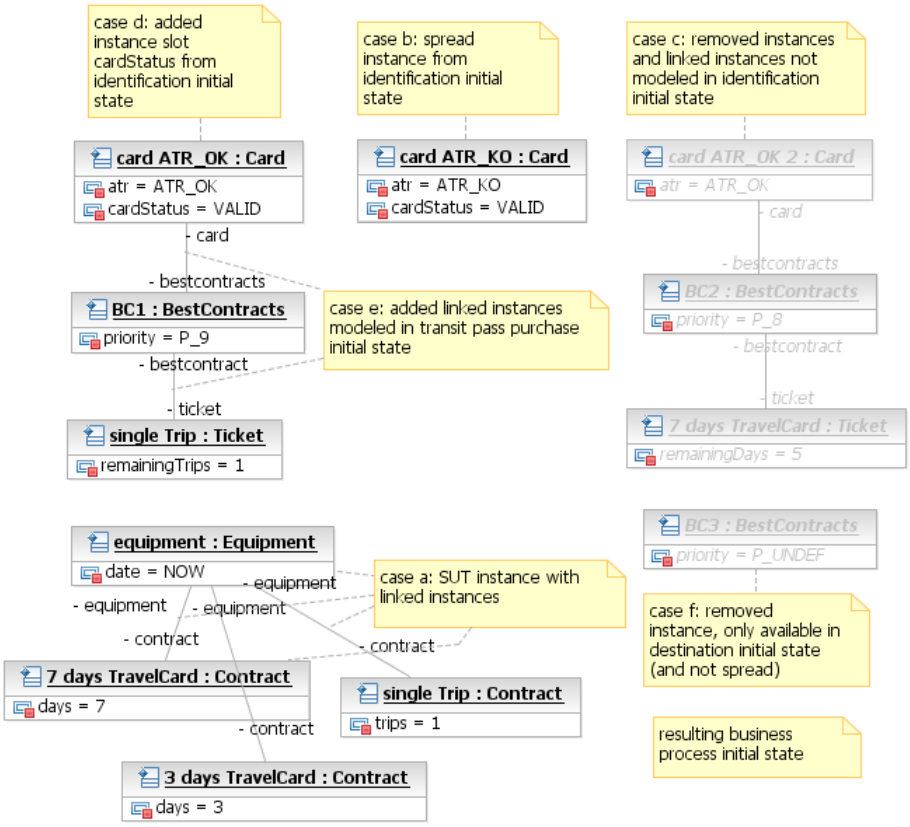


Figure 11. *Resulting initial state diagram*

– If all **values are equal in every initial states**, we keep those values.

– **If that's not the case**, we must specify how to decide value to be retained. Non instantiated attributes in source initial states will be given an arbitrary value during tests concretization, thus will not appear in resulting initial state.

– **Attributes existing in several diagrams** but with only one significative value and all other not instantiated, will remain in resulting initial state with the significative value.

– In the worst case, that is **multiple significative values different in all diagrams**, we have to choose the most significative value in the business process context. This choice can only be made by a domain expert.

We now have a system under test class instance that we should be completed with every linked instances available in initial states, that will be added to the resulting initial state (Figure 11 case a).

Classes of spread instances must not have any setter (an operation that will lead a generator to automatically defines instances values). Calls to those operations will canceled any modeling effort made in business process activity diagram to underline spread instances. Those specific instances have to be handled differently. We must take into account both source and destination initial states (instances are spread from one activity to an other, i.e. a machine to an other as shown in Figure 6). There are three alternative :

1) All instances available in source state are directly used in resulting state (Figure 11 case b).

2) An instance of destination initial state not present in source initial state is not used in resulting state, excluding also linked instances (Figure 11 case c).

3) An instance available in both source state and destination state is used in resulting state with values extracted from initial state (Figure11 case d).

Links between instances are preserved as expressed in source initial state, eventually augmented with links modeled in destination initial state. A comprehensive navigation through diagrams links is mandatory to ensure a complete instances processing (Figure 11 case e). This implies instances functional coherence in object diagrams.

All other classes, not modeled in business process activity diagram, are insignificant in this processing and those instances are not retain in resulting initial state (Figure 11 case f).

3.2. *Dealing with control nodes*

There are two types of control nodes in activity diagrams : fork and decision. We focus on these two types of control nodes. The other two ie merge and join have no impacts on the creation of the initial state. Both have restrictions and characteristics that have to be examined when building initial state.

Through a fork node, an activity starts two concurrent control flows. Objects can pass from activity to activity within those parallel flows. If we take a look at object spreading from a parallel portion to a non parallel one, we identify two cases :

– *two control flows (or more) and two (or more) instances of* ***different classes*** is the trivial case, all instances are used in resulting state.

– *two control flows (or more) and instances of the* ***same class***. This case is only possible when dealing with different instances of the same class. The same instance cannot be used by two parallel flows, that would create a non deterministic model useless for tests generation.

A decision node introduces at least two possible control flows. Just like fork node, a case where different instances spread from a branch to the main control flow, is quite simple. Instances from source initial state are used in resulting initial state. But if an instance spread from one or more branches to the main one, and has equal attributes

values whatever the branches, this instances will be added to resulting initial state. Different attributes values implies that this instance will be cloned into resulting state. A new clone will be created for each new value. Then we will add all links and linked instances available in diagrams.

By the end of this composition phase, we have added to the model, a new state machine, made of sub-machines, a class diagram and a new initial state. All those information will be used in combination with a class diagram, to generate new tests suites that cover business process.

4. Business process test suites generation

In one hand, Business process test generation can be done directly from the expression of this business process such as expressed in (Z. J. Li *et al.*, 2008), where BPEL is used to describe the process. A specific test generation tool is able to cover execution path and produce non-regression tests. In the other hand, we can a UML model. We decide to reuse UML model produced during earlier phases. We can generate tests the same way we would have processed a single state machine describing a simple behavior and using model-based testing. We used Smartesting Test Designer to identify targets from the model and generate tests to cover business process functional behaviors. The tool unfolds the new state machine augmented with the new initial state. We thus obtain logical tests that must be concretized to be executable. It is the publication phase that will transform our logical tests into Junit or Fitness tests. The published tests have to be automated. To do so control points are implemented. Then mapping between the data of the initial state or the verdict must be done to interact with the application to be tested. Thus each state machine should have tests suites to singly guarantee its coherence in regard with requirements. We introduce a new set of tests ensuring a good integration of those separate modules when used by a business process. We produce new integration tests that should not be confused with tests produced to maintain functional coverage.

Business process generated tests offer the benefit of covering a larger functional pane, taking into account data that are known to be business specific. Unreachable targets can be caused by the new initial state, this reveals that a business process may not use every module functional behaviors. Other unreachable targets caused by data exchange between activities can be detected.

Table 1 presents the results of test generation for the business process shown in Figure 12. We can see the 3 simple behaviors and results in terms of requirements, targets and generated tests. For the identification activity, the tool identifies 17 targets corresponding to transitions to activate the state machine. It generates 11 tests to cover these 17 targets and 8 requirements involved. For the 3 simple behaviors we have the same number of tests as we did when generating them independently. The interest lies in the use of a common initial state. The other strong point is the modeling of 3 new behaviors introduced by the user's type dealing with the system. We produce 3 tests, 2

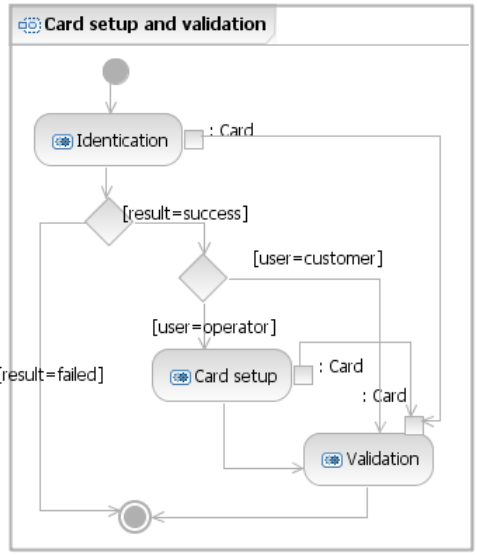


Figure 12. *Card setup and validation business process*

Behaviours	Requirements	Targets	Tests
Card Identification	8	17	11
Card setup	2	2	2
Validation	14	31	20
Business process	0	3	3

Tableau 1. *Results on tests suites generation*

are new tests and the third one is a test from *Identification* failed activity. We can also note that there were no business requirements expressed in the specification, and only recovered requirements from previous activities are available.

A great advantage of this method is to easily produce runnable tests, and though we do not have to deal with combinatorial explosion. That would be the case, if we were automatically connecting every exit points with every entry points of our activities. This would be an interesting feature for robustness testing, but we only want to offer more functional testing of a system. And a better way to achieve it, is to give the opportunity to a business analyst to express useful business behaviours. We produce integration tests considering we are composing behaviours, in addition with data navigating from one activity to another. These data represent a major interest because they allow to define new use case, and ensure a better integration of components with each other. Another advantage is the use of the new initial state by tests already produced

at the activities level. The 33 tests may use any available initial state data. Therefore, some behaviours may not be accessible with these data that can be explained by the fact that a business process does not activate every behaviours available.

5. Conclusion

This work is directly linked to Smartesting technology but can be easily applied to other techniques such as (Briand *et al.*, 2001) and (Nebut *et al.*, 2006). Model-based testing methodology and technology play a large role to ease functional validation of developments, with a permanent concern for software quality improvements. We have introduced in this paper the possibility to model and to use business processes to carry-on with those improvements. They have a great added-value in the model, as they describe new functional behaviors. They also supplement the model with a new level of abstraction, allowing a better model structuring. Methodology presented here was put in practice on a concrete example with industrial partners. However a problem resides in resulting machine depth. Even if tests generation is feasible for each sub-machine composing the final machine, there is no certainty that generation will be possible for the resulting machine.

Expressing a business process in the correct notation was part of our research. Several languages can be found to deal with business processes description : UML, BPMN or DFD. Both notation can be used for this task depending on criteria or measurement grids, and evaluation can be found in (Nysetvold *et al.*, 2005, Wahl *et al.*, 2005, Wohed *et al.*, 2006, Recker *et al.*, 2005). Those languages are related because they were created or improved to deal with the same task : modeling a business process. The main difference resides in the final user qualification. BPMN was created to give business professional a notation to produce models. UML was introduced as an answer to modeling standardization of software development. Activity diagrams were updated in UML 2.0 to deal with business processes modeling specific needs. However UML is still an adequate notation to describe technical domain (White, 2004) and is our final choice for the project.

One key point of this work is to offer a methodology to design business processes from existing model elements. This methodology is a complement to the previous one extending the Unified Process to produce models suitable for tests generation. We now have a complete methodology to achieve functional testing of software applications. A business process is regarded as an integration test, and automatically generated tests augment campaigns already produced by classical model-based testing. Process may be modeled by business expert allowing introduction of information not yet identified. Thus the expert becomes a validation team member and gives a new point of view on the model and on developed functional aspects. A strong point of the methodology is the ability to identify data spreading between activities and to use them for tests generation. Thereby a process without any data spreading would have no added value.

In the same vein, we wish to emphasize the reuse of information from the model. For software product lines it is typically the case. A model will be used for all new versions of application, this may be true for the model to the test. A model previously produced and a specific methodology can lead to a substantial gain in time and money.

6. Bibliographie

Arlow J., Neustadt I., *UML 2 and the Unified Process, Second Edition, Practical Object-Oriented Analysis and Design*, Addison-Wesley, 2003.

Bouquet F., Debricon S., Legeard B., Nicolet J.-D., « Extending the Unified Process with Model-Based Testing », *MoDeVa'06, 3rd Int. Workshop on Model Development, Validation and Verification*, Genova, Italy, p. 2-15, October, 2006.

Bouquet F., Grandpierre C., Legeard B., Peureux F., Utting M., Vacelet N., « A subset of precise UML for Model-based Testing », *A-MOST'07, 3rd Workshop on Advances in Model Based Testing*, London, UK, July, 2007. Proceedings on CD. A-MOST'O7 is colocated with ISSTA 2007, Int. Symposium on Software Testing and Analysis.

Briand L. C., Labiche Y., « A UML-Based Approach to System Testing », *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, Springer-Verlag, London, UK, p. 194-208, 2001.

Nebut C., Fleurey F., Traon Y. L., Jézéquel J.-M., « Automatic Test Generation : A Use Case Driven Approach », *IEEE Trans. Software Eng.*, vol. 32, n° 3, p. 140-155, 2006.

Nysetvold, Krogstie, « Assessing Business Process Modeling Languages Using a Generic Quality Framework », *10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'05)*, Porto, Portugal, 2005.

Recker J. C., Indulska M., Rosemann M., Green P., « Do Process Modelling Techniques Get Better ? A Comparative Ontological Analysis of BPMN », *16th Australasian Conference on Information Systems*, Sydney, Australia, 2005.

Utting M., Legeard B., *Practical Model-Based Testing - A tools approach*, Elsevier Science, 2006a. 550 pages, ISBN 0-12-372501-1. To Appear.

Utting M., Pretschner A., Legeard B., « A Taxonomy of Model-Based Testing », *Technical report 04/2006, Department of Computer Science, The University of Waikato (New Zealand)*, April, 2006b.

Wahl, Sindre, « An Analytical Evaluation of BPMN Using a Semiotic Quality Framework », *10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'05)*, Porto, Portugal, 2005.

White S. A., Process Modeling Notations and Workflow Patterns, Technical report, IBM Corp., 2004. http ://www.bpmn.org/Documents/Notations and Workflow Patterns.pdf.

Wohed P., van der Aalst W. M., Dumas M., ter Hofstede A. H., Russell N., « On the Suitability of BPMN for Business Process Modelling », *Proceedings 4th International Conference on Business Process Management 4102*, Vienna, Austria, p. 161-176, 2006.

Z. J. Li H. F. Tan H. H. L. J. Z., Mitsumori N. M., « Business-process-driven gray-box SOA testing », *IBM Systems Journal*, 2008.