IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 4, No 9, July 2010
ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

Verifying ODP trader function by using Event B

Belhaj Hafid, Bouhdadi Mohamed, El hajji Said

**Department of Mathematics & Computer Science, University Mohammed V, Faculty of science
BP 1014 RP, 4. Av Ibn Batouta – Agdal, Rabat, Morocco**

Abstract
In order to support interoperability in open distributed systems, an information service is needed that can provide dynamic knowledge about available service providers. Such a service is Trading function, identified by Basic Reference Model of Open Distributed Processing (RM ODP). RM ODP is a joint effort of ISO and ITU−T. Within the standardization of RM ODP, Trading function is developed as a component standard.
The use of formal methods in the design process of ODP systems is explicitly required. Currently there are no formal specifications of ODP concepts which are widely accepted. One interesting question concerns the suitability of event B for their use in ODP. In this paper, the use of event B for verifying ODP is investigated and evaluated. The ODP trader is chosen as case of study because it appears as a first main application of ODP.
Keywords: *RM-ODP, Trader function, event B, RODIN platform.*

1. Introduction

One property of a distributed system is that a user of the system is unaware of the differences in computers and operating systems in which their applications run. Such systems are inherently complex. Despite this, distributed processing is growing rapidly, primarily due to the computer industry's ability to produce cheaper, more powerful computers. As a result of this growth, the need for the coordinated production of standards for distributed processing has been identified.
ODP is already a major effort between the International Organization for Standardization (ISO) and International Telecommunications Union (ITU-T) which will lead to significant product development in the coming years. The ODP work identifies and attempts to provide a framework for distributed systems. This has been set out in a Reference Model of ODP (RMODP) [1].
It defines an architecture through which distribution, interworking and portability can be achieved. The RM-ODP recognizes that it cannot provide an infrastructure to meet all of the needs of distribution. Different systems will almost certainly have different demands on the infrastructure.
The RM-ODP is divided into four main parts.
Part 1 - Overview and Guide to Use : contains an overview and guide to use of the RM-ODP.
Part 2 - Descriptive Model : contains the definition of concepts and gives the framework for descriptions of distributed systems.
Part 3 - Prescriptive Model : contains the specification of the required characteristics that qualify distributed system as open, i.e. constraints to which ODP systems must conform. It defines a framework comprising five viewpoints, five viewpoint languages, ODP functions and ODP transparencies. The five viewpoints are enterprise, information, computational, engineering and technology.
Part 4 - Architectural Semantics : contains a formalization of a subset of the ODP concepts.
A trader [3] is an object that performs trading, which is an ODP common function. ODP aims to provide distribution-transparent utilization of services over heterogeneous environments. In order to use services, users need to be aware of potential service providers and to be capable of accessing them. Since sites and applications in distributed systems are likely to change frequently, it is advantageous to allow late binding between service users and providers. If this is to be supported, a component must be able to find appropriate service providers dynamically. The ODP trading function [3] provides this dynamic selection of service providers at run time.
The languages Z, SDL, LOTOS, and Esterel are used in RM-ODP architectural semantics part [1] for the specification of ODP concepts. However, no formal method is likely to be suitable for specifying every aspect of an ODP system.
Elsewhere, we used the meta-modeling approach [9] [10] to define syntax of a sub-language for the ODP QoS-aware enterprise viewpoint specifications. We defined a meta-model semantics for structural constraints on ODP enterprise language [11] using UML and OCL. We also used the same met-modeling and denotation approaches for behavioral concepts in the foundations part and in the enterprise language [12] [13].
Furthermore, for modeling ODP systems correctly by construction, the current testing techniques [21][22] are not widely accepted.
In this paper, we use the event-B formalism as our formal framework for developing trader function in distributed

systems. Event B [4] is a method with tool support for applying systems in the B method. Hence we can benefit from the useful formalism for reasoning about distributed systems given by refinement techniques and from the tool support in B. The Rodin Platform for Event-B provides effective support for refinement and mathematical proof. [5]
The paper is organized as follows. Section 2 introduces RM-ODP and trader function. Section 3, presents an introduction to event B notations. In Section 4, we use event B as refinement support to specify trader function. Section 5 presents the Rodin platform as tool of proving initial and refinement models. In section 6 we describe related works. Lastly, section 7 concludes the paper.

2. Introduction to ODP and the RM-ODP

2.1 RM-ODP

The Reference Model for Open Distributed Processing (RM-ODP) [1] provides a framework within which support of distribution, networking and portability can be integrated. It consists of four parts. The architecture part contains the specifications of the required characteristics that qualify distributed processing as open. It defines a framework comprising five viewpoints, five viewpoint languages, ODP functions and ODP transparencies. The five viewpoints are enterprise, information, computational, engineering and technology. The ODP functions are required to support ODP systems. The transparency prescriptions show how to use the ODP functions to achieve distribution transparency. RM-ODP defines a number of specific repository functions [2], concerned with maintaining a database of specialized classes of information. There are three repository functions: Type Repository, Relocator and the trader function [3].

2.2 Trader Overview

A trader [3] is a third party object that enables the clients to find suitable servers in a distributed system. Figure 1 shows the interactions of a trader and its users:
• A trader accepts service offers from exporters of services when exporters wish to advertise service offers. A service offer contains the characteristics of a service that a service provider is willing to offer. Service offers are stored by the trader in a centralized or a distributed database.
• A trader accepts service requests from importers of services when importers require knowledge about appropriate service providers.
• A trader searches its service offer database to match the importer's service request. And, if required, a trader can select the most appropriate service offer(s) (if one exists) that satisfies the importer's service request. The matched list of service offers or the selected service offer is returned to the importer.
After a successful match, the client, that requires a service, can interact with the service provider of a matched offer. The matching and selection of appropriate service at run time by a trader allows client objects to be configured into an ODP system without prior knowledge of server objects that can satisfy their requirements.

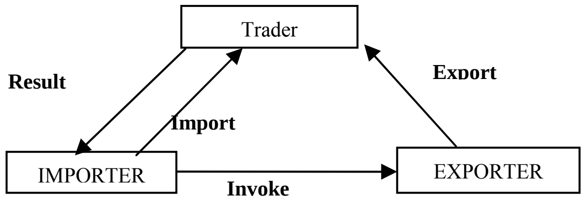


Fig. 1 Trader and Its Users.

3. EVENT B MODELLING APPROACH

The Event-B [14] [15] is formal techniques consist of describing rigorously the problem, introduce solutions or details in the refinement steps to obtain more concrete specifications and verifying that proposed solutions are correct. The system is modeled in terms of an abstract state space using variables with set theoretic types and the events that modify state variables. Event-B, a variant of B, was designed for developing distributed systems. In Event-B, the events consist of guarded actions occurring spontaneously rather then being invoked. The invariants state properties that must be satisfied by the variables and maintained by the activation of the events.
The mathematical foundations for development of event based system in B is discussed in [6]. An abstract machine consists of sets, constants and variables clause modeled as set theoretic constructs. The invariants and properties are defined as first order predicates. The event system is defined by its state and contain number strained by the conditions defined in the properties and invariant clause known as invariant properties of the system. Each event in the abstract model is composed of a guard and an action. A typical abstract machine may be outlined as below.

```
MACHINE              M
SETS                 S1,S2,S3...
CONSTANTS            C
PROPERTIES           P
VARIABLES            v1,v2,v3...
INVARIANTS           I
INITIALISATION       init
EVENTS
      E1 = WHEN  G1 THEN  S1   END;
      .......                  END
```

4. SPECIFYING THE ODP TRADER USING EVENT B

4.1 Refinement strategy

In this section, we present our strategy for constructing the negotiation scenario between trader object and its users. This will be done by means of an initial model followed by one refinement.

a) The initial model essentially presents the protocol as done abstractly of repository function of trader.

b) In the refinement model, we introduce the trader repository function. A trader needs to checks with its type repository that the service type specified in the offer is valid.

4.2 Abstract Model of trading function

The abstract model of a trader communication protocol is presented as a B machine in the Fig. 2. The PROCESS and MESSAGE are defined as sets. The brief description of this machine is given as follows.

```
MACHINE        Trader and users communication
SETS      PROCESS ={importer, exporter, trader)
          MESSAGE = {import, export, result, invoke}
VARIABLES  sender , receiver
INVARIANT
/* I-1*/     sender  ∈  MESSAGE ⇸  PROCESS
/* I-2*/   ∧  receiver ∈  PROCESS ↔ MESSAGE
/* I-3*/   ∧  ran(receiver) ⊆  dom(sender)
INITIALISATION  sender := Φ || receiver:= Φ
OPERATIONS
Send ( pp ∈  PROCESS , mm ∈  MESSAGE) =
 WHEN  mm ∈  dom(sender) ∧  pp ∈  dom(receiver)
  THEN  sender := sender U {mm ↦ pp}
  END;
Receive (pp ∈  PROCESS , mm ∈  MESSAGE) =
 WHEN mm ∈  dom(sender)  ∧  (pp ↦ mm) ∈  receiver
  THEN  receiver := receiver U {pp ↦ mm}
END ;
END
```

Fig. 2 Abstract Model of trader function.

The sender is a partial function from MESSAGE to PROCESS defined in invariant I-1. The mapping $(m \mapsto p) \in$ sender indicate that message m was sent by process p. The receiver is a relation between PROCESS and MESSAGE defined in invariant I-2. A mapping of form $(p \mapsto m) \in$ receiver indicates that a process p has delivered a message m. The sender and receiver are initialized as empty set.

In our model of trader communication protocol with its users, a sent message is also delivered to its sender. It may be noticed that all delivered messages must be messages whose Message Sent event is also recorded. This property is defined as invariant I-3. The events of sending and receiving of messages are modeled as Send(pp, mm) and Receive(pp, mm). When a Send event is invoked, the entry of a process and the corresponding message is made to the sender.

4.3 Refinement model: Introducing repository function of a trader

In order to match service requests with service offers, a trader interacts with the type repository function provided by the ODP infrastructure [2]. The set of all service types known to a trader is known to its type repository.

The possible interaction scenario for the trader and its environment is given below:

Interaction 1. Service Export - the trader receives a service offer from an exporter. The trader checks with its type repository that the service type (or interface type), the service properties and service offer properties specified in the offer are valid. The service offer is stored in the trader database including the offer's service type identifier (if given), interface type identifier, service and service offer properties.

Interaction 2. Service import - the trader receives a service request from a client. The trader checks with its type repository that the request contains a known service or interface type and the properties in the matching constraints are valid.

Interaction 3. Matched offers - the trader returns offers (possibly empty) to the importer that matches the importer's requirement specifications.

Finally, to use the service, the importer needs to map the service interface identifier to an interface location for the service, establish a binding with the server at the service location and, finally, invoke the service.

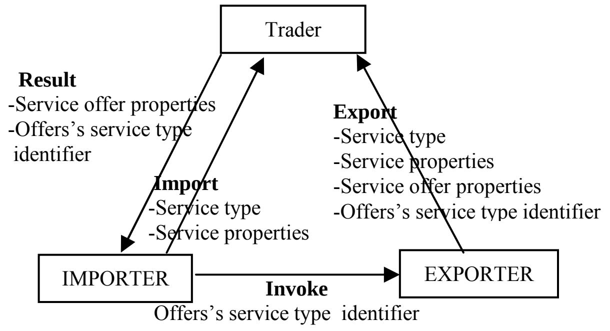

Trader
Result
-Service offer properties
-Offers's service type identifier
Export
-Service type
-Service properties
-Service offer properties
-Offers's service type identifier
Import
-Service type
-Service properties
IMPORTER
EXPORTER
Invoke
Offers's service type identifier

Fig. 3 Trader and Its Users taking in account the repository function.

In this refinement we introduce the repository interface of trader. The refinement of abstract model is given in Fig. 4 and Fig. 5. A brief description of the refinement steps are given below.
The abstract repository interface is represented by a variable repository. A mapping of the form (m1 ↦ m2) ∈ repository indicates that parameter m1 is sent with message m2 (Inv I-4). A repository on the messages can be defined only on those messages whose message sent event is recorded (Inv I-5).

REFINEMENT Repository interface
REFINE Trader and users communication
SETS PROCESS = {importer, exporter, trader);
MESSAGE = {import, export, result, invoke};
PARAMETER= {Service type, Service properties, Service offer properties, Offers' service type identifier}
CONSTANT Trader type repository
VARIABLES Sender, receiver, repository
INVARIANT
/* I-4* repository ∈ PARAMETER ↔ MESSAGE
/* I-5*/ ∧ ran (repository) ⊆ dom (sender)
INITIALISATION sender := Φ || receiver := Φ || repository := Φ

Fig. 4 Trader communication with users and repository: Initialization

The events send (pp, mm, param) and receive (pp, mm, param) respectively models the events of sending a message and the receiving of a message.
As shown in the operations, only exporter process can export a service offer. When the trader receives a service offer from an exporter pp, the trader checks with its type repository (Trader type repository) that the service type is valid. The service offer is stored in the trader database including the offer's service type identifier, interface type identifier, service and service offer properties.
Furthermore, only importer process can import a service. When the trader receives a service request from a client, the trader checks with its type repository (Trader type repository) that the request contains a known service or interface type and the properties in the matching constraints are valid.

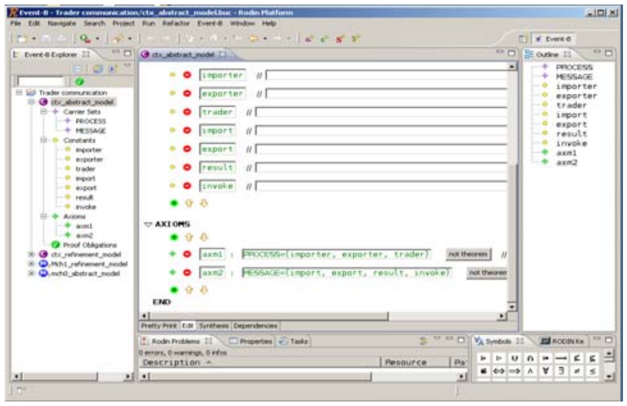
Fig. 5: Trader communication with users and repository: Events

OPERATIONS
Send (pp ∈ PROCESS, mm ∈ MESSAGE, param ∈ PARAMETER) =
WHEN mm ∈ dom (sender) ∧ pp ∈ dom (receiver) ∧ param ∈ dom(repository)
∧ pp =exporter
∧ Service_type = Trader_type_repository
THEN sender := sender U {mm ↦ pp} || repository := repository U {param ↦ mm}
END;
Receive (pp ∈ PROCESS , mm ∈ MESSAGE, param ∈ PARAMETER) =
WHEN mm ∈ dom(sender) ∧ pp ∈ dom(receiver) ∧ param ∈ dom(repository)
∧ pp = importer
∧ Service_type = Trader_type_repository
THEN receiver := receiver U {pp ↦ mm} || repository := repository U {param ↦ mm}
END ;

Fig. 5 Trader communication with users and repository: Events

5. PROOFING TRADER MODELS

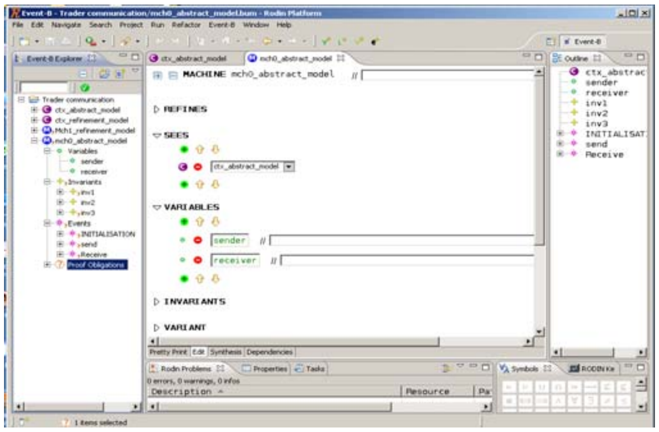
Rodin Platform [5] is an open tool set implemented on top of Eclipse. It is devoted to supporting the development of such systems. It has been developed within the framework of the European project Rodin. It contains a modeling database surrounded by various plug-ins: static checker, proof obligation generator, proovers, model-checkers, animators, UML transformers, requirement document handler, etc. The database itself contains the various modeling elements needed to construct discrete transition system models: essentially variables, invariants, and transitions.
The initial model of trader communication with users and its refinements models are developed by using Event-B. Each model was analyzed and proved to be correct using The Rodin Platform. The correctness of each step is proved in order to achieve a reliable protocol communication between trader, client and server objects.
The abstract and refinement models of the trader by both essentials construct of Event-B (machine and context) are illustrated below:



importer //
exporter //
trader //
import //
export //
result //
invoke //
AXIOMS
axm1 : PROCESS={importer, exporter, trader} not theorem //
axm2 : MESSAGE={import, export, result, invoke} not theorem
END

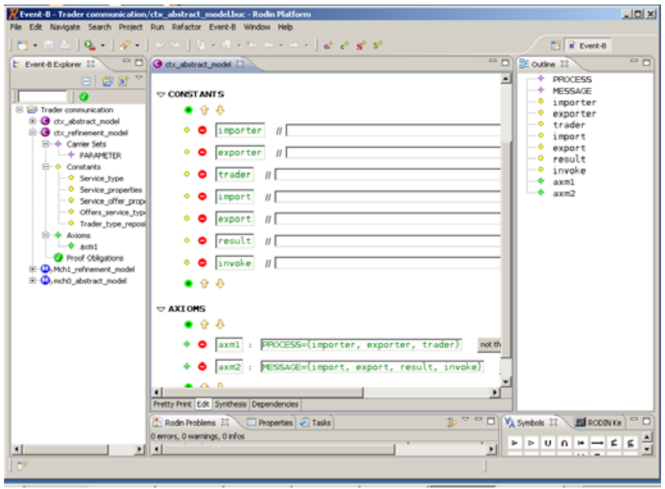
0 errors, 0 warnings, 0 infos

Fig. 6 A context of trader abstract model.



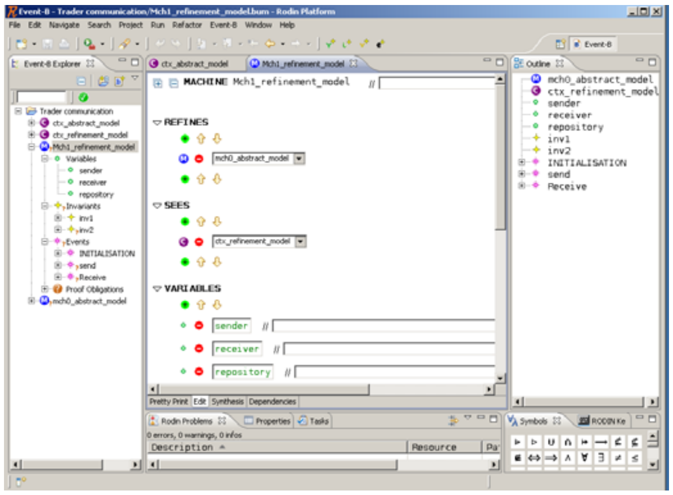
```
MACHINE mch0_abstract_model
REFINES
SEES
    ctx_abstract_model
VARIABLES
    sender
    receiver
INVARIANTS
VARIANT
```

Fig. 7 A machine of trader abstract model.



```
CONSTANTS
    importer
    exporter
    trader
    import
    export
    result
    invoke
AXIOMS
    axm1 : PROCESS={importer, exporter, trader}
    axm2 : MESSAGE={import, export, result, invoke}
```

Fig. 8 A context of trader refinement model.



```
MACHINE Mch1_refinement_model
REFINES
    mch0_abstract_model
SEES
    ctx_refinement_model
VARIABLES
    sender
    receiver
    repository
```

Fig. 9 A machine of trader refinement model.

6. RELATED WORK

There are related works which are using event-B to specify systems. For example, Abrial [8] introduces patterns for state-based specifications in EventB and uses informal graphical notations similar to TD to illustrate the patterns. Cansell et al. [16] introduce a time constraints pattern based on an Event-B model for distributed applications. This work uses global time which interacts with a number of active times as do our patterns. Bicarregui [17] extends Event-B notations to three linear temporal logic (LTL) operators The work proposes using three new constructs that are to replace the standard Event-B structure, WHEN…THEN…END, to represent the three LTL operators. In [18] the use of the Formal Description Techniques (FDT's) Z, LOTOS and SDL'92 is investigated and evaluated for specifying the ODP trader. KAOS [19] is a goal-oriented modeling technique for requirements specification, in which a goal defines an objective of the composite system. KAOS uses a Goal model to declare the system requirements. An attempt to combine KAOS with B is introduced by Ponsard and Dieul [20]. Our earlier works [11] [23] investigates how to translate the specification of enterprise viewpoint concepts in event-B. Our work is unique in providing techniques for verifying ODP trader specification by using the standard Event B notations provided.

7. CONCLUSION AND PERSPECTIVES

In this paper we have presented a formal approach to modeling and analyzing trading function using Event B. The abstract model of trader is done abstractly of its repository function.
In the refinement of the abstract model, we introduced the notion of a trader repository function. In fact, in order to match service requests with service offers, a trader interacts with the type repository function provided by the ODP infrastructure. The set of all service types known to a trader is known to its type repository.
The system development approach considered is based on Event B, which facilitates incremental development of distributed systems. The work was carried out on the Rodin platform. In order to verify our trader model, the initial and refinement model of trader are developed by using Event-B, Each model is analyzed and proved to be correct.
Our experience with this case study strengthens our believe that abstraction and refinement are valuable technique for modeling complex distributed system.
As for future work, we are going to generalize our approach to verify ODP common function trader from different viewpoint. This will be our basis for further investigation of using event-B in the design process of ODP systems

References

[1] ISO/IEC, ''Basic Reference Model of Open Distributed Processing-Part1, 2,3 ,and 4: 'ISO/IEC 1994

[2] ISO/IEC, "ODP Type Repository Function", ISO/IEC JTC1/SC7 N2057, 1999. 24. ISO/IEC, "The ODP Trading Function", ISO/IEC JTC1/SC21 1995.

[3] ITU/ISO " ODP Trading Function - Part 1 ; Specification", ISO/IEC IS 13235-1, ITU/T Draft Rec X950 - 1, (1997)

[4] http://www.event-b.org/

[5] RODIN. Development Environment for Complex Systems (Rodin). 2009. http://rodin.cs.ncl.ac.uk/.

[6] J.-R. Abrial. The B-Book: Assigning programs to meanings. Cambridge University Press, 1996.

[7] J.-R. Abrial. Tools for Constructing Large Systems (a proposal). In Rigorous Development of Complex Fault-Tolerant Systems. M. Butler, etc. (Eds). LNCS 4157 Springer, 2006

[8] J.-R. Abrial, Tutorial - Case study of a complete reactive system in Event-B: A mechanical press controller. Proc. 5th International Symposium on Formal Methods (FM'2008), Turku, Finland, 2008.

[9] M. Bouhdadi et al., ''A UML-Based Meta-language for the QoS-aware Enterprise Specification of Open Distributed Systems'' IFIP Series, Vol 85, Springer, (2002) 255-264.

[10] Mohamed Bouhdadi and Youssef Balouki. 'A Semantics of Behavioral Concepts for Open Virtual Enterprises'. Series: Lecture Notes in Electrical Engineering, , Vol. 27 .Springer, 2009. p.275-286.

[11] H. Belhaj and al. Event B for ODP Enterprise Behavioral Concepts Specification, Proceedings of the World Congress on Engineering 2009 Vol I, WCE '09, July 1 - 3, 2009, London, U.K., Lecture Notes in Engineering and Computer Science, pp. 784-788, Newswood Limited, 2009

[12] Youssef Balouki and Mohamed Bouhdadi. 'Using BPEL for Behavioral Concepts in ODP Enterprise Language', Virtual Enterprises and Collaborative Networks, IFIP, Vol. 283, pp. 221-232, Springer, 2008

[13] Mohamed Bouhdadi and Youssef Balouki and El maati Chabbar, 'Meta-modeling Syntax and Semantics of Structural Concepts for Open Networked Enterprises', Lecture Notes in Computer Science, Vol. 4707, pp. 45-54, Springer, 2007.

[14] Joochim, T., Snook, C., Poppleton, M. and Gravell, A. (2010) TIMING DIAGRAMS REQUIREMENTS MODELING USING EVENT-B FORMAL METHODS. In: IASTED International Conference on Software Engineering (SE2010), February 16 – 18, 2010, Innsbruck, Austria.

[15] C.Snook & M.Butler, UML-B and Event-B: an integration of languages and tools. Proc. IASTED International Conf. on Software Engineering (SE2008), Innsbruck, Austria, 2008.

[16] D. Cansell, D. Méry & J. Rehm, Time Constraint Patterns for Event B Development. Proc. Formal Specification and Development in B, 7th International Conf. of B (B 2007), Besancon, France, 2007. 140-154.

[17] J. Bicarregui, et al, Towards Modeling Obligations in Event-B. Proc. International Conf. of ASM, B and Z Users, London, UK, 2008, 181-194.

[18] Joachim Fischer , Andreas Prinz and Andreas Vogel, Different FDT's confronted with different ODP-viewpoints of the trader. Book Series Lecture Notes in Computer Science Éditeur Springer Berlin / Heidelberg Volume 670/1993, Pages 332-350.

[19] E. Letier & A.V. Lamsweerde, Agent-Based Tactics for Goal-Oriented Requirements Elaboration. Proc. 24th International Conf. on Software Engineering (ICSE'02), Orlando, Florida, USA, 2002, 83-93.

[20] C. Ponsard & E. Dieul, From Requirements Models to Formal Specifications in B. Proc. International Workshop on Regulations Modeling and their Validation and Verification (REMO2V'06), Universitaires de Namur, Luxemburg , 2006, 249-260.

[21] Myers, G. The art of Software Testing, John Wiley &Sons, New York, 1979

[22] Binder, R. Testing Object Oriented Systems. Models. Patterns, and Tools, Addison-Wesley, 1999

[23] Hafid Belhaj, Youssef Balouki, Mohamed Bouhdadi, Said El hajji: Using Event B to specify QoS in ODP Enterprise language. PRO-VE'10 11th IFIP Working Conference on VIRTUAL ENTERPRISES, Saint-Etienne, France, 11-13 October 2010. (accepted)

Belhaj Hafid is a PhD student in Computing sciences at the University of Rabat. He received his MSc in Computer Science (Architecture of Information Systems and Communication) from National Engineer School of Computer Science and System Analysis (ENSIAS), Rabat, Morocco. Topic: Conception and development of a platform for decisional reactive agents. Since 2001 he was an engineer in Moroccan Pension Fund (C.M.R) overseeing it design, development and information architecture.
He has experience in teaching, research, and student supervision in software engineering and logic programming. His main research interest is in the formal specification of open distributed systems and model driven architecture.