IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 2, March 2011
ISSN (Online): 1694-0814
www.IJCSI.org

From UML Activity Diagrams to CSP Expressions: A Graph Transformation Approach using Atom³ Tool

Raida Elmansouri[1], Houda Hamrouche[2] and Allaoua Chaoui[1]

[1] MISC Laboratory, Department of Computer Science, University Mentouri Constantine, Constantine 25000, Algeria

[2] Department of Computer Science, University of Skikda, Skikda, Algeria

Abstract

The Unified Modeling Language (UML) has become a widely accepted standard in the object oriented software development industry. However, the UML is a semi-formal language which lacks precisely defined constructs. On the other hand, CSP language is a formal specification language. So, UML and CSP have complementary features: UML can be used for modeling while CSP can be used for analysis. In this paper we propose an approach and a tool to transform UML activity diagrams to CSP. Our approach is based on graph transformation and uses ATOM^3 tool. The purpose of this transformation is to provide some verification of properties ranging from simple deadlock verification to more specific properties.

Keywords: *UML activity diagram, CSP, Graph transformation, ATOM^3*

1. Introduction

The Unified Modeling Language (UML) [2] has become a widely accepted standard in the object oriented software development industry. Some diagrams are used to model the structure of a system while others are used to model the behavior of a system. UML Statecharts and UML collaboration diagrams are widely used to model the dynamic behavior in UML. UML State chart diagrams model the lifetime (states life cycle) of an object in response to events. A UML collaboration diagram models the interaction between a set of objects through the messages (or events) that may be dispatched among them. *Activity diagrams* are used to model workflow systems, service oriented systems and business processes. Control flow includes support for sequential, choice, parallel and events. Activities may be grouped in sub-activities and can be nested at different levels. However, the UML is a semi-formal language which lacks rigorously defined constructs.

Communicating Sequential Processes (**CSP**) [3] is a formal language for describing patterns of interaction in concurrent systems. It is a member of the family of mathematical theories of concurrency known as process algebras, or process calculi. CSP was first proposed in 1978 by C. A. R. Hoare, but has since evolved substantially. CSP has been practically applied in industry as a tool for specifying and verifying the concurrent aspects of a variety of different systems.

So, UML and CSP have complementary features: UML can be used for modeling while CSP can be used for analysis. Much research has been done about the integration of UML and CSP. In [8], the authors present a case study of UML activity diagram to CSP transformation using graph transformation. In [9], the authors describe how an UML activity diagram can be transformed into a corresponding CSP expression by using the graph rewriting language PROGRES. In [6], the authors exploited ATOM^3 [1] [4] for transforming UML statecharts diagrams and collaboration diagrams to Colored Petri nets [5].

In this paper we propose an approach and a tool for transforming UML activity diagrams to CSP expressions. The main difference between our approach and the previously cited approaches consists in the fact that we use ATOM^3 as graph transformation tool.

The rest of this paper is organized as follows. In section 2, we review the main concepts of UML activity diagrams, CSP, and graph transformation. In section 3, we describe our approach that transform a UML activity diagram to CSP code. In section 4, we illustrate our approach using an example. The final section concludes the paper and gives some perspectives.

2. Background

2.1 UML activity diagrams

Activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structured development. Figure 1 represents an example of an activity diagram that depicts one way to model the logic of the *Enroll in University* use case. *Fill Out Enrollment Forms*, *Enroll in University* are examples of activities. For more details, the reader is referred to [Booch99].

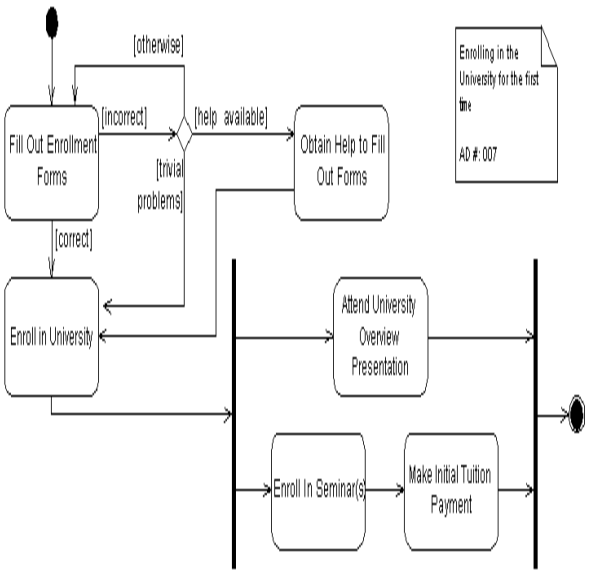

Fig. 1 Example of activity diagram

2.2 CSP

The theory of CSP itself is also still the subject of active research, including work to increase its range of practical applicability. CSP allows the description of systems in terms of component processes that operate independently, and interact with each other solely through message-passing communication. However, the *"Sequential"* part of the CSP name is now something of a misnomer, since modern CSP allows component processes to be defined both as sequential processes, and as the parallel composition of more primitive processes. The relationships between different processes, and the way each process communicates with its environment, are described using various process algebraic operators. Using this algebraic approach, quite complex process descriptions can be easily constructed from a few primitive elements. For more details, the reader is referred to [3].

2.3 Graph Transformation

Graph grammar [7] is a generalization of Chomsky grammar for graphs. It is a formalism in which the transformation of graph structures can be modeled and studied. The main idea of graph transformation is the rule-based modification of graphs as shown in Figure 3.

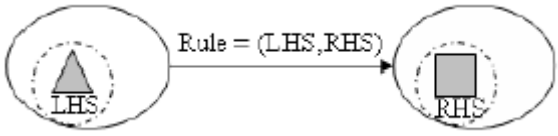

Fig. 2 Rule-based Modification of Graphs

Graph grammars are composed of production rules; each having graphs in their left and right hand sides (LHS and RHS). Rules are compared with an input graph called host graph. If a matching is found between the LHS of a rule and a subgraph in the host graph, then the rule can be applied and the matching sub-graph of the host graph is replaced by the RHS of the rule. A rewriting system iteratively applies matching rules in the grammar to the host graph until no more rules are applicable.

ATOM^3 [1] is a visual tool for multi-formalism modeling and meta-modeling. The two main tasks of AToM^3 are meta-modeling and model transformation. Meta-modeling refers to modeling formalism concepts at meta-level, using Entity Relationship (ER) formalism or UML Class Diagram formalism extended with the ability to express constraints. Once we build the meta-models for the interested models, ATOM^3 can generate automatically a visual modeling tool. For Model transformation, AToM^3 supports graph rewriting system, which uses graph Grammar rules to visually guide the procedure of the transformation. For more details, the reader is referred to [9].

3. The Approach

The proposed approach consists of transforming a UML activity diagram to CSP. To reach this

objective, we have proposed a meta-model for UML activity diagram and a graph grammar that performs automatically the transformation of a UML activity diagram. In the following, we give in details these two steps.

3.1 Meta-Modeling Activity diagrams

To Meta-model activity diagrams, we proposed the meta-model below containing seven classes linked by thirty three associations as shown in figure 3.

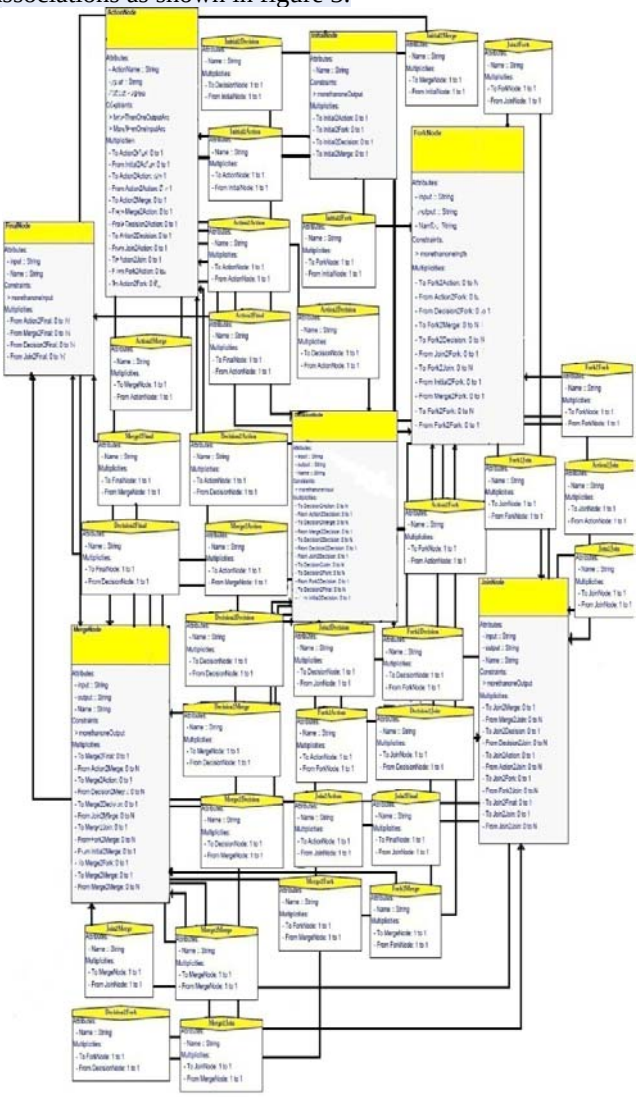

Fig. 3 Activity diagram Meta-Model

Each association of this meta-model has an attribute of type String. It links an instance of the class with a single source instance of the Class Destination. Cardinalities for each association are:

- To target: 1 to 1.
- From source: 1 to 1.

Some classes are described as follows:

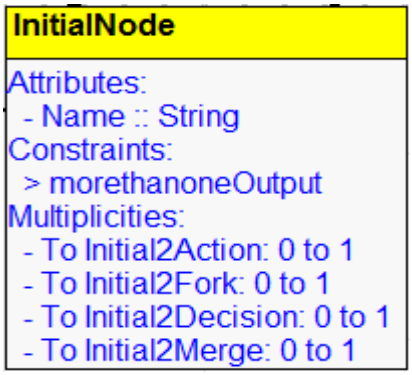

Fig. 4 InitialNode Class

1. **InitialNode Class:** represents the beginning of an activity diagram. Graphically it is represented by a small solid circle. It has a constraint which prohibits the existence of more than one outgoing arc. It can be connected by one of the following associations:
   - ***Initial2Action*** : with the class *ActionNode.*
   - ***Initial2Decision*** : with the class *DecisionNode.*
   - ***Initial2Fork*** : with the class *ForkNode.*
   - ***Initial2Merge*** : with the class *MergeNode.*

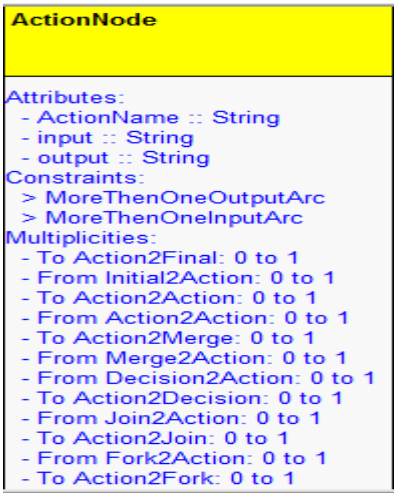

Fig. 5 ActionNode Class

Generation of a tool for Activity diagrams

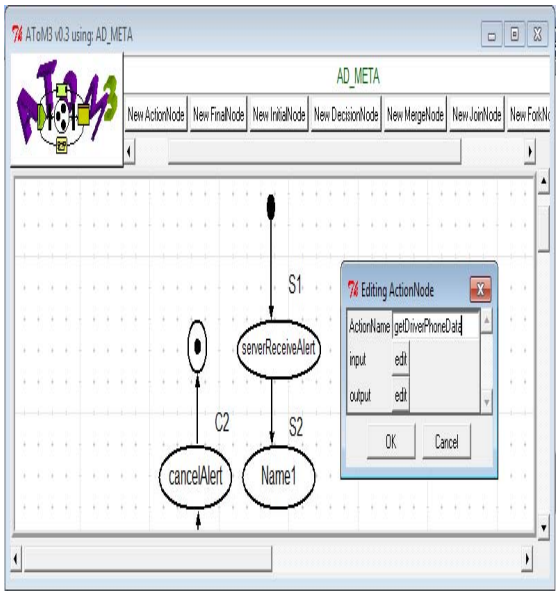

Fig. 7 Tool generated to model the Activity diagrams

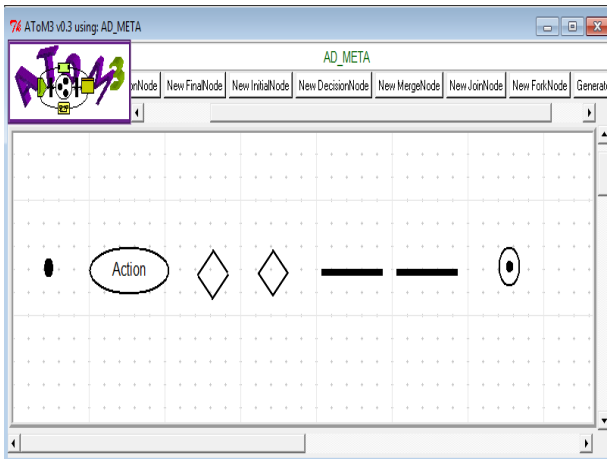

Fig. 8 Graphical representation of the nodes of activity diagrams

This figure graphically represents the nodes of activity diagrams modeled by our tool. From left to right are: an initial node, a node of action, a decision node, a node fusion, a node bifurcation, a common node and *end* node.

Graph grammar for the transformation of UML activity diagrams to CSP expressions

To generate CSP expressions from a UML activity diagram, we have proposed Thirty nine rules. For lack of space we only describe in the following some rules.

- ***Rule 1 : rule_Initial2Action (Priority 1) :***
  This rule is applied to locate a node of action previously untreated, can be connected to the initial node by an incoming labeled arc. The name of the latter is assigned to the attribute 'input' node of action will then be marked as "Visited" for the first time.

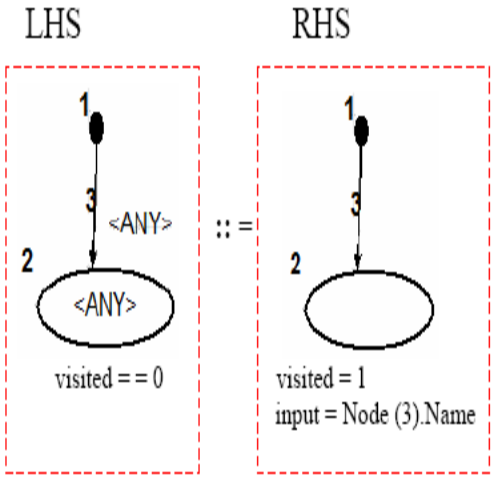

Fig. 9 LHS and RHS of grammar rule 1

- ***Rule 2 : rule_ Action2Action (Priority 2) :***
  Used to locate two nodes connected by an arc action, whose destination is not yet processed. The name of the arc is kept as "output" at the source node, and as "input" at the destination node. This is marked as "Visited" for the first time.

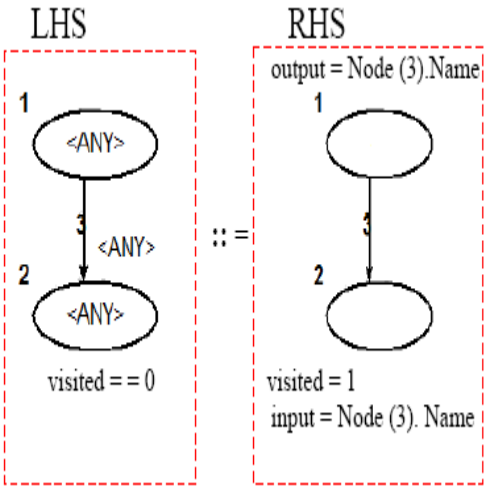

Fig. 10 LHS and RHS of grammar rule 2

- ***Rule 3 : rule_ Action2Final (Priority 3) :***
  Applied to locate an arc previously untreated, connecting a node of action (source) to an end node (destination). The name of this arc is

assigned as the "output" of the source node and is concatenated with "input" of the destination node.

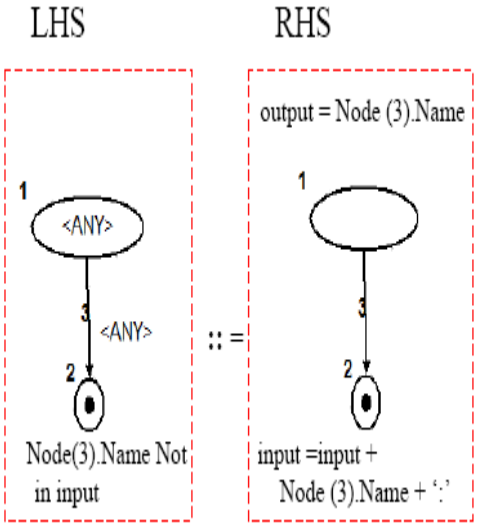

Fig. 11. LHS and RHS of grammar rule 3

- ***Rule 4: rule_ Merge2Final (Priority 4) :*** Applied to locate an arc previously untreated, connecting a node merge (source) to an end node (destination). The name of this arc is assigned to "output" of the source node, and concatenated with "input" of the destination node.

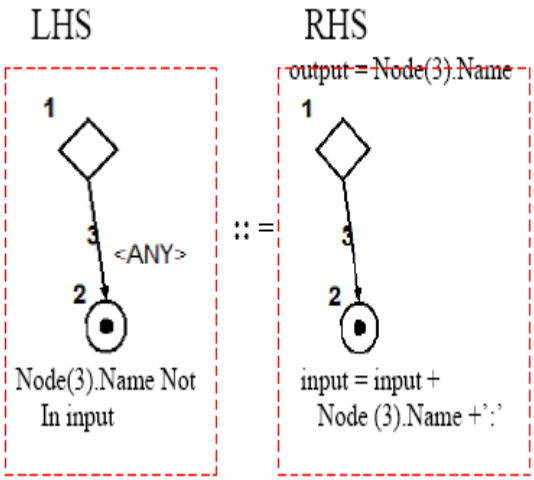

Fig.12 LHS and RHS of grammar rule 4

3 Example

We have applied our approach on the activity diagram of figure 13, and obtained automatically the equivalent CSP code of figure 15. An intermediate step of the application of our approach is shown in figure 14.

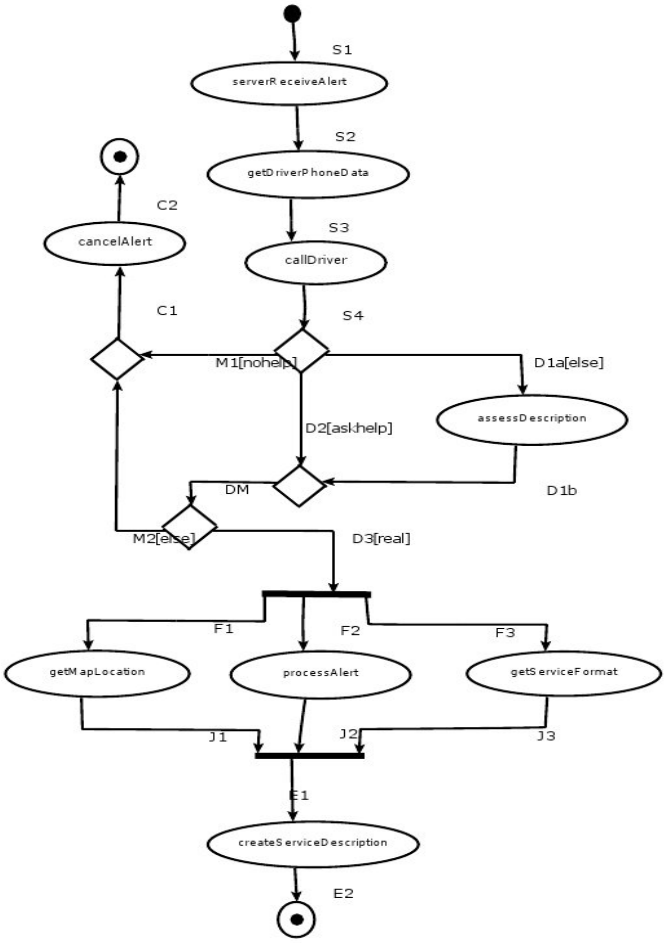

Fig. 13 Example of an activity diagram created by our tool

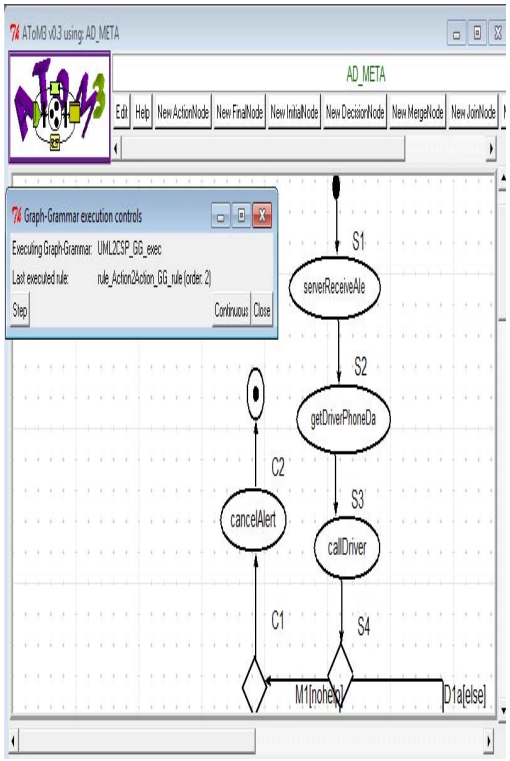

Fig.14 An Intermediate step of execution of the graph grammar

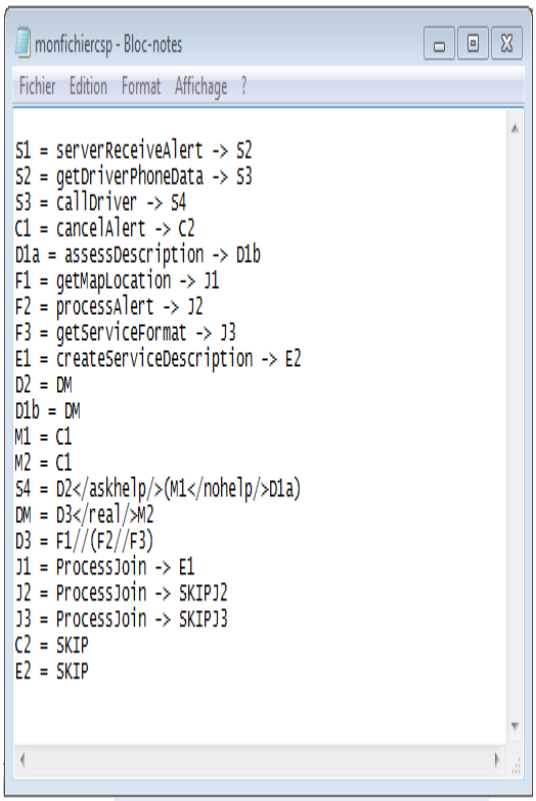

Fig.15 CSP code equivalent to the activity diagram

shown in Figure 13.

4 Conclusion

In this paper we proposed an approach and a tool for transforming UML activity diagrams to CSP code. Our approach is based on graph transformation and uses ATOM^3 tool. In a future work, we plan to transform other UML diagrams to CSP expressions. We plan also to perform some verification of properties using CSP.

References

[1] AToM^3 Home page, version 3.00, http://atom3.cs.mcgill.ca/

[2] G. Booch, Ivar Rumbaugh and Jim Jacobson: The Unified Modeling Language
User Guide, Addison-Wesley, 1999.

[3] C.A.R. Hoare, Communicating Sequential Processes. Prentice Hall International Series in Computer Science. Prentice Hall (April 1985)

[4] J. De Lara and H. Vangheluwe: "AToM^3: A Tool for Multi-Formalism Modeling and Meta-Modeling". LNCS 2306, pp.174-188. Presented also at Fundamental Approaches to Software Engineering - FASE'02 , in European Joint Conferences on Theory And Practice of Software - ETAPS'02, Grenoble, France, 2002.

[5] K. Jensen: Coloured Petri Nets, Vol 1: Basic Concepts, Springer-Verlag 1992.

[6] E. Kerkouche, A. Chaoui, E. Bourennane, O. Labbani. A UML and Colored Petri Nets Integrated Modeling and Analysis Approach using Graph Transformation. In Journal of Object Technology, vol. 9, no. 4, 2010, pages 25–43. Available at http://www.jot.fm/contents/issue_2010_07/article2.html

[7] G. Rozenberg: Handbook of Graph Grammars and Computing by Graph Transformation, World Scientific, 1999.

[8] D. Bisztray, K. Ehrig, and Reiko Heckel. Case Study: UML to CSP Transformation. Available at http://www.informatik.uni-marburg.de/~swt/agtive-contest/UML-to-CSP.pdf

[9] E. Weinell and U. Ranger**.** Using PROGRES for Transforming UML Activity Diagrams into CSP Expressions. Available at www.se.rwth-aachen.de/files/agtivetc/**UML**_to_**CSP**.pdf

Raida Elmansouri is with the department of computer science, Faculty of Engineering, University Mentouri Constantine, Algeria. She received her Master degree in Computer science in 1997 and her PhD degree in 2009 from the University of Constantine. Her field of interest include information systems and formal methods.

Houda Hamrouche is a Master student at the University of Skikda in Algeria.

Allaoua Chaoui is with the department of computer science, Faculty of Engineering, University Mentouri Constantine, Algeria. He received his Master degree in Computer science in 1992 (incooperation with the University of Glasgow, Scotland) and his PhD degree in 1998 from the University of Constantine (in cooperation with the CEDRIC Laboratory of CNAM in Paris, France). He has served as associate professor in Philadelphia University in Jordan for five years and University Mentoury Constantine for many years. During his career he has designed and taught courses in Software Engineering and Formal Methods. Dr Allaoua Chaoui has published many articles in International Journals and Conferences. He supervises many Master and PhD students. His research interests include Mobile Computing, formal specification and verification of distributed systems, and graph transformation systems.