Component-oriented Software Development with UML

Nara Sueina Teixeira
Graduate Program in Computer Science
Federal University of Santa Catarina - UFSC
Florianópolis, Brasil
E-mail: narasueina@inf.ufsc.br

Ricardo Pereira e Silva
Department of Informatics and Statistics - INE
Federal University of Santa Catarina - UFSC
Florianópolis, Brasil
E-mail: ricardo@inf.ufsc.br

Abstract— **This paper proposes to automate the process of structural and behavior analysis of component-oriented software fully specified in UML. The structural specification uses component, class and deployment diagrams, and the behavior specification, state machine diagram. The produced structural analysis tool analyzes a connection between pairs of components at a time. The produced behavioral analysis tool considers the behavior of the system as a whole, leading to behavioral specification of the application automatically from the machine state of each connected component. It is performed the convertion of the state machines of the individual components and of the application to Petri nets in a transparent manner to the user. The behavioral assessment is done by analyzing Petri net properties, considering the context of the components. Analysis results are produced without demand effort, allowing early location of design problems.**

Keywords-Component-oriented development; structural compatibility analysis; behavioral compatibility analysis; UML; Petri Nets.

I. INTRODUCTION

For the component-based software development approach, software construction consists in an interconnection of a collection of units: the components. “A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment” [1]. “A component interface (CI) is a collection of service access points, each one with a defined semantics” [2]. The latter establishes the services required and provided by a component, not considering implementation details.

Some research efforts suggest the automation of component compatibility analysis evaluating their CIs. Dias and Vieira [3] use the Argus-I tool integrated to the SPIN tool [4] for the component compatibility analysis, in which specifications are produced in ADL (Architecture Description Language) and state machine diagram is converted to PROMELA [5]. The architectural analysis considers the "super state model", but the authors do not detail how it is generated.

Chouali and Souquières [6] use refinement in B to prove the compatibility between two interfaces, through the tool AtelierB [7]. The CI specification is converted to the formal method B and consists of a data model associated with each component provided and the required interface. The interoperability does not cover behavioral aspects, therefore it does not assess the feasibility of the component-based application.

Mouakher, Lanoix and Souquières [8] improved the approach [6] by adding an interface protocol, described in PSM (Protocol State Machine), to the CI specification and proposing adapters when incompatible interfaces were identified. However, the analysis is also performed between two connected interfaces, disregarding problems associated with the whole set of application components. In [6] and [8] the notion of component port is not treated.

Bracciali, Brogi and Channel [9] describe the interface of components through IDLs (Interface Description Language) and they use a subset of Lambda Calculus to represent the behavior of components. This low level solution becomes difficult to be applied to describe complex systems.

The component compatibility analysis should be performed based on the CI specification and must consider three distinct aspects: structural, behavioral, and functional. “The structural aspect concerns the static features of a component and corresponds to the set of required and provided operation signatures of the CI. The behavioral aspect defines constraints in the invocation order of provided and required operations. The functional aspect describes what the component does, not necessarily going into details of its implementation” [10].

The lack of a widely accepted standard for the specification of CI makes the analysis of compatibility between components difficult and hence, their reuse. The second version of UML, called henceforth UML [1] provides mechanisms to deal with components, but does not establish a standard for complete specifications.

In a previous publication [11] were proposed ways of specifying component-oriented software and CI, in which the specification is based on the object-oriented paradigm and uses only UML diagrams. For the CI structural specification component and class diagrams are used and for the CI behavioral specification is utilized the state machine diagram. Thus, each component has its own state machine (SM) representing its externally observable behavior – being this observable behavior the sequence of required and provided operations performed during the component’s operation. The organization of components of an application is described by using the deployment diagram.

This paper proposes the automation of the component’s compatibility analysis process from the component-based software specification [11]. The approach used in this paper is implemented in the current version of the SEA environment [10] [12] [13], which uses UML. SEA is a development environment in which the object-oriented paradigm is used for production and use of reusable software artifacts. Some tools were built in this environment to automate the analysis of structural and behavioral compatibility.

The structural analysis tool (SAT) handles at each time, pairs of connected ports. The behavioral analysis tool (BAT) considers not only the individual behavior of each component, but also the behavior of the system as a whole. It involves the entire set of application's interconnected components at the same time. In this work, the application SM is automatically obtained from the union of the SMs of each connected component.

For the behavioral analysis, the UML state machine diagram is converted in Petri net (PN). This conversion is done automatically in a transparent way to the user, who does not need any knowledge of this modeling technique. Behavioral problems are identified through the interpretation of PN properties, considering the component context. The conversion method (of SM to PN) used in this study is similar to that proposed in [14], however, it only handles PNs of the ordinary kind and presents particularities of the treaty context.

Functional compatibility analisys consists in evaluate if the execution steps of an operation are in agreement with the need of the component that invokes the operation. This kind of analisys is not automatable and is beyond the scope of this work.

The following sections are organized as: Section II presents concepts related to OCEAN / SEA, and Section III presents the approach to specify component-based software. In Section IV, the automated structural compatibility analysis is described, while in Section V, the behavioral analysis is presented. Software specification and analysis are supported by the tools inserted in SEA environment. Section VI presents how the evaluation of the produced tools occurred. The article ends with conclusions, in Section VII.

II. OCEAN/SEA IMPLEMENTATION

OCEAN [10] is an object-oriented framework for the domain of the software development environments. From this framework, SEA environment, a software development support, was built.

The software development using SEA starts with the production of a UML design specification. In this environment, a design specification is an object that aggregates models and concepts (that are objects) and includes relationships between these objects. Each kind of UML diagram is defined as a class related to the proper diagram elements, that is, to the classes that model the diagram elements.

In the SEA environment, tools are also defined as classes and they are related to one or more kinds of specification – the ones that can be handled by these tools. The tools can be produced to be accessed by a menu or to be automatically called in a specific situation.

Tools of an OCEAN-based environment are produced by means of framework extension (subclassing). There are three kinds of tool: editors (such as a diagram editor), converters (such as a code generator), and analyzers. The analyzers read a design specification without changing it and produce reports with the specification features. The tools SAT and BAT are analyzers.

III. SPECIFICATION OF COMPONENT-BASED SOFTWARE

A. *Structural Specification*

The structural specification concerns to all the operation signatures of the CI. "The CI refers to the portion of the component responsible for communicating with its external environment. Taking into account the nomenclature of UML, the CI is composed by a port collection, each one associated to one or more UML interfaces" [11].

In this approach, producing the CI structural specification requires the specification of all interfaces associated with the component (in class diagram) and the definition of the component ports, associating required or provided interfaces to each of them in component diagram.

With the establishment of the interfaces related to the component ports through realization or dependency relationship, it becomes possible to check what operations are provided or required from a component's port.

Figure 1 illustrates the structural specification of a hypothetical component, CompB, made in the SEA environment. At the right side there is the class diagram with the interfaces; at the left side, a component, in a component diagram, that is related to the declared interfaces.

In the SEA environment, the connection between components is made in the deployment diagram, linking the ports of connected components. Figure 2 illustrates a deployment diagram with a hypothetical software artifact consisting of the interconnection of three components. All the components must be declared in component diagram and all the interfaces, in class diagram.

B. *Behavioral Specification*

The CI behavioral specification sets restrictions on the invocation order of operations provided and required by the component. In this approach, the behavioral specification is represented by a UML state machine diagram. The basic idea is that each state represents a situation that occurs during the operation of a component, which is characterized by the operations required and provided that can be performed at the time. Each transition leaving a state represents the execution of an operation – provided or required – that can leave the component in the same state or lead to another state. Some conventions have been established:

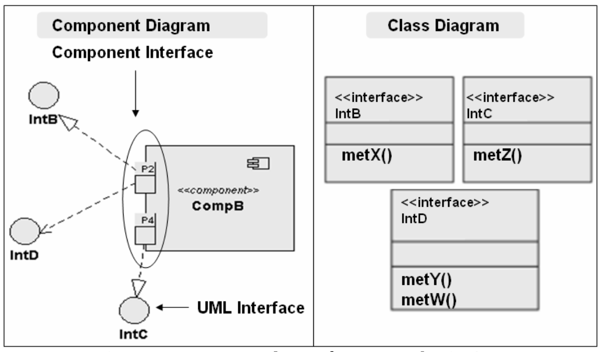


Figure 1. Component structural specification in the SEA environment.

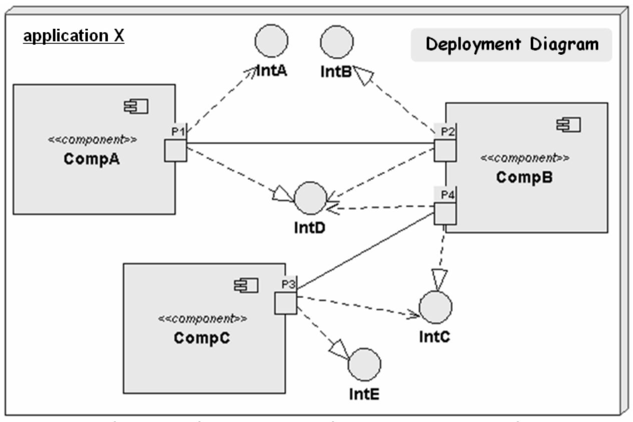


Figure 2. Software artifact consisting of the interconnection of components CompA, CompB, CompC.

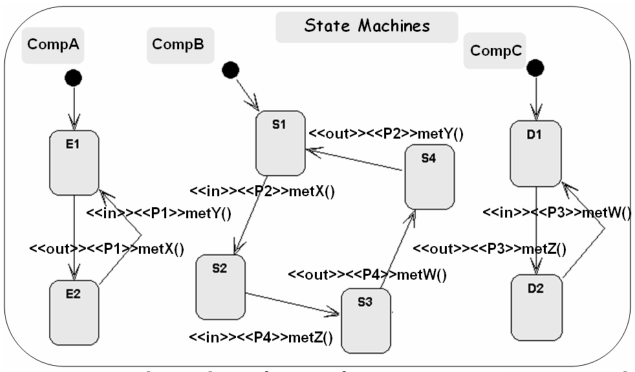


Figure 3. Behavioral specification of components CompA, CompB and CompC.

- The state identifiers are combinations of letters and numbers, which only differentiate a state of the others (the transitions are the elements that define the semantics of the model).

-The transitions are labeled according to the following convention: <direction> <port> <operation>, where <direction> may be <<out>> for the operations invoked by the component and <<in>> for provided operations.

Figure 3 illustrates the SMs of the components CompA, CompB and CompC (mentioned in Figure 2).

IV. Automation of Structural Components's Compatibility Analysis

Figure 4 illustrates the SAT performance. Its purpose is to perform structural analysis, which consists in the following actions:

A. *Structural Specification Consistency Analysis*

The structural specification consistency analysis verifies if the system is specified with all restrictions set forth in approach, such as:

- All components are specified in a component diagram with at least one port associated to each one.

- Each port is associated with at least one required or provided interface.

- Each interface referenced in the component diagram is described in a class diagram.

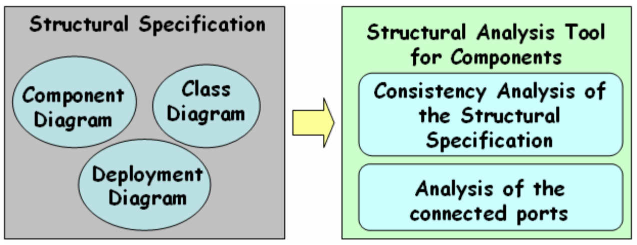

Structural Specification
Component Diagram
Class Diagram
Deployment Diagram
Structural Analysis Tool for Components
Consistency Analysis of the Structural Specification
Analysis of the connected ports

Figure 4. Structural Analysis Tool of the SEA environment (SAT).

- Each interface defined in the class diagram has at least one declared operation.

- At least two components are connected in a deployment diagram.

B. *Connected Port Analysis*

The structural compatibility is evaluated for each pair of connected ports of the application components. "The set of required operations by a port includes the operations of all interfaces related to that port by dependency. These operations should be provided by the port on the other side of the connection through its set of provided operations, in other words, the set of operations of all interfaces related to that port by realization" [11]. Otherwise, structural incompatibility is identified in the connection.

The analysis of the connected ports compares, for each pair of connected ports, the operations required in the port of a component with the operations provided by the port of the other component attached to it, considering operation name, return type, number of parameters and parameter type.

At the end of the analysis, SAT reports the results, with the structural incompatibilities found.

V. Automation of the Component Behavioral Compatibility Analysis

Figure 5 illustrates the BAT operation in the SEA environment. In this approach, the behavioral analysis of components involves the following actions:

A. *Behavioral Specification Consistency Analysis*

The behavioral specification consistency analysis checks whether the specification complies with the restrictions established in the approach, such as those mentioned in Section III-B.

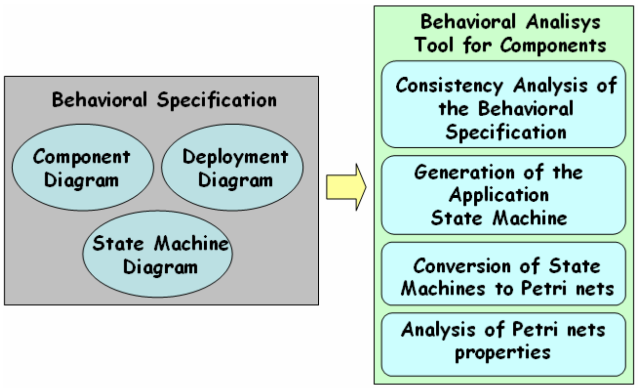

Behavioral Specification
Component Diagram
Deployment Diagram
State Machine Diagram
Behavioral Analisys Tool for Components
Consistency Analysis of the Behavioral Specification
Generation of the Application State Machine
Conversion of State Machines to Petri nets
Analysis of Petri nets properties

Figure 5. Behavioral Analysis Tool of the SEA environment (BAT).

For each analyzed SM, a behavioral specification evaluation report is generated, assessing the respective component, with the found errors.

B. Generation of the application behavioral specification

"Behavioral compatibility is observed between components if the restrictions on the operation invocation order of the required and provided operations established in each component are compatible with the other components connected to it. This type of evaluation involves the whole set of connected components" [11].

The method for generating the application SM was proposed in a previous work [11] and consists of:

1. Identifying pairs of related transitions. Two transitions are related if they involve interconnected ports and execution of the same operation, which is required on one side and provided by the other side;

2. Inserting fork and join pseudostates (a single syntactic element) that synchronizes the related transitions of the different machines. This link will convert the set of SMs in a single one – the component-based application SM – and synchronizes an operation invocation with its execution;

3. Synchronize the transitions of the initial pseudostates of various machines with a fork pseudostate (inserting a single initial pseudostate for the application SM). This step preserves the initial state of all SMs.

From this algorithm, the application SM will include the states of all involved components. Figure 6 illustrates the SM (automatically generated by BAT) of the application, consisting of the interconnection of components CompA, CompB and CompC, illustrated in the Figures 2 and 3.

C. Conversion of State Machines in Petri Nets

The user of the SEA environment manipulates only UML diagrams to specify component-based software. The SMs are converted into the corresponding PNs automatically, in a completely transparent way to the user, who never see PN diagrams.

The algorithm for conversion of the SMs in PNs is summarized in the following steps:

1. For each state of the SM, create a place in the PN.

2. Identify the states related to the initial pseudostate and mark the corresponding places with a token at each PN.

3. For each SM transition not related to another, create a transition and connect it with arcs to its input and output places (it applies to the SM transitions of the individual components and the application SM transitions corresponding to unconnected ports).

4. For each set of SM transitions related to a fork/join in the application, create a transition with a set of arcs connecting it to their respective input and output places.

Figure 7 illustrates the PN obtained from the conversion of the SM showed in Figure 6.

D. Petri net properties analysis

The Pipe analyzer tool – Platform Independent Petri net Editor 2, version 2.5 [15] – was integrated to the SEA environment, with adaptations and extensions. Given a PN, the analyzer, through state enumeration, reports whether or not it has a certain property. The interpretation of each property is done for the treated context. The following properties are considered:

1. Safeness: the PNs that represent component-based applications must be safe. Otherwise, it denotes behavioral error.

2. Reversibility: in this study, the conclusion that a PN that represent component-based applications is not reversible causes a warning which should be evaluated by the user.

3. Deadlock: a deadlocked PN characterizes a behavioral error. This can occur for two reasons: one is because the restrictions associated to the execution order of the operations, established by a component, are not respected by other components connected to it. Another reason is the occurrence of unconnected port(s) in one or more application components. It occurs when the component requires or provides operations, through this port, which are essential to its operation.

4. Liveness: an alive PN representing a component-based application characterizes a behavioral specification without errors. However, the absence of this property does not necessarily denote behavioral error. A not alive PN may have almost alive or dead transitions and is the user's responsibility to assess whether or not this is a behavioral compatibility problem.

5. Almost alive transitions: this characteristic leads to a warning, because it is necessary that the user evaluates if the unavailability of an operation, at a certain moment of the execution, is a behavior compatibility problem.

6. Dead transitions: this feature also requires the user evaluation, that is, if the permanent unavailability of an operation is a problem for the application.

7. Transition invariants: In the analysis of the application PN, the invariants are identified and compared with the invariants of the individual component PNs, because possible cyclic sequences of operations of a component may not be possible when it is connected to others.

The analysis of the PN properties is made for both application PN and individual component PNs. The interpretation that occurs to this case is the same as the application PN, except for the property deadlock: considering the specification of an individual component, deadlock means modeling inconsistency. It is necessary to compare situations that occur in the component behavior, but that no longer occur in the application, when the component is connected to others.

VI. PROPOSED APPROACH EVALUATION

Two emphases have been adopted in the evaluation process: the tools's ability to identify errors and suspicious situations (reported as warnings) and the appropriateness of the analysis approach. The evaluation of the implemented tools was performed with small applications, with a maximum of ten components. Specifications without errors and specifications with purposely inserted errors were treated by the analysis tools in order to evaluate all situations in which they should work.

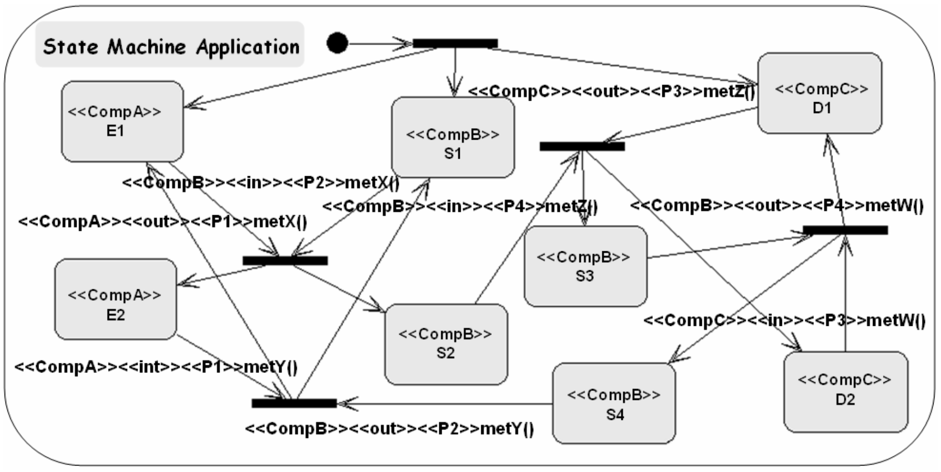

State Machine Application
<<CompA>> E1
<<CompB>> S1
<<CompC>> D1
<<CompC>><<out>><<P3>>metZ()
<<CompB>><<in>><<P2>>metX()
<<CompB>><<in>><<P4>>metZ()
<<CompB>><<out>><<P4>>metW()
<<CompA>><<out>><<P1>>metX()
<<CompA>> E2
<<CompB>> S3
<<CompB>> S2
<<CompC>><<in>><<P3>>metW()
<<CompA>><<int>><<P1>>metY()
<<CompB>> S4
<<CompC>> D2
<<CompB>><<out>><<P2>>metY()

Figure 6. Behavioral specification corresponding to the application of the figure 2.

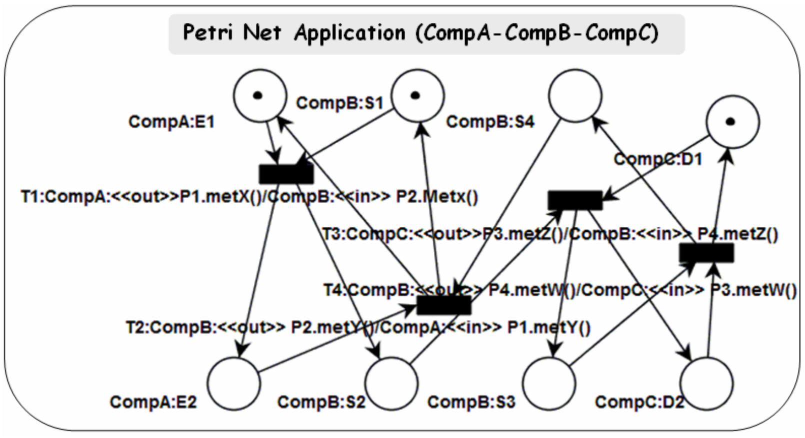

Petri Net Application (CompA-CompB-CompC)
CompA:E1
CompB:S1
CompB:S4
CompC:D1
T1:CompA:<<out>>P1.metX()/CompB:<<in>> P2.Metx()
T3:CompC:<<out>>P3.metZ()/CompB:<<in>> P4.metZ()
T4:CompB:<<out>> P4.metW()/CompC:<<in>> P3.metW()
T2:CompB:<<out>> P2.metY()/CompA:<<in>> P1.metY()
CompA:E2
CompB:S2
CompB:S3
CompC:D2

Figure 7. Petri net obtained from the conversion of the SM showed in Figure 6.

The analysis approach showed to be adequate when comparing their results with the conclusions of not automatic analysis. The tools were not submitted to stress test. The following are some analysis examples.

Figure 8 shows an exaple of a structural analysis report with error detection – in this case, operation not provided and problems with parameteres. Figure 9 shows an example of behavioral analysis report with errors due to unconnected port, that is, a deadlock caused by the need of operation invocation in an unconnected port. Figure 10 shows another example of behavioral analysis report with warnings due the possible changes that may occur in the component behavior when it becomes part of a component connection. In this case, possible service execution cycles of an individual component not be preserved when it is connected to other components. Besides that, operations always available in the components become temporarily unavailable in the application that contains the components.

Based on reports like the ones showed, the user can make decisions and define corrective action related to the component and application specifications. For situations that represent warnings, the user must evaluate whether or not they mean a problem for the application.

VII. Conclusion and Future Works

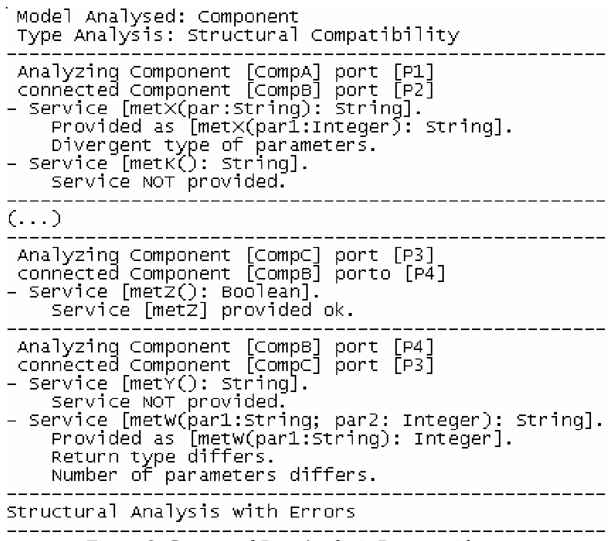
This paper has presented an automatic procedure for the structural and behavioral compatibility analysis. The approach was implemented in the SEA environment, using tools embedded in it.

Component and class diagrams have been used for the CI structural specification. The behavioral aspect is defined using the state machine diagram. The component organization is defined using the deployment diagram.

The structural analysis tool evaluates whether the operations required on one side of the connection are provided by the component on the other side.

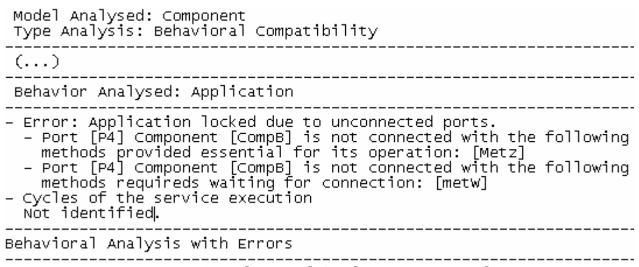
The behavioral analysis tool generates the application SM automatically. All SMs are converted into PNs, which are analyzed and interpreted in the given context.

The main advantages of this proposal are: the specification is made with just a single language, UML; the application behavioral specification is generated automatically, reducing the design effort; the behavioral analysis considers the behavior of individual components and application, comparing them and identifies errors and suspicious situations reported as warnings.



```
 Model Analysed: Component
 Type Analysis: Structural Compatibility
-----------------------------------------------------
 Analyzing Component [CompA] port [P1]
 connected Component [CompB] port [P2]
- Service [metX(par:String): String].
    Provided as [metX(par1:Integer): String].
    Divergent type of parameters.
- Service [metK(): String].
    Service NOT provided.
-----------------------------------------------------
(...)
-----------------------------------------------------
 Analyzing Component [CompC] port [P3]
 connected Component [CompB] porto [P4]
- Service [metZ(): Boolean].
    Service [metZ] provided ok.
-----------------------------------------------------
 Analyzing Component [CompB] port [P4]
 connected Component [CompC] port [P3]
- Service [metY(): String].
    Service NOT provided.
- Service [metW(par1:String; par2: Integer): String].
    Provided as [metW(par1:String): Integer].
    Return type differs.
    Number of parameters differs.
-----------------------------------------------------
Structural Analysis with Errors
-----------------------------------------------------
```

Figure 8. Connected Port Analysis Report with error.



```
 Model Analysed: Component
 Type Analysis: Behavioral Compatibility
--------------------------------------------------------------------
 (...)
--------------------------------------------------------------------
 Behavior Analysed: Application
--------------------------------------------------------------------
- Error: Application locked due to unconnected ports.
  - Port [P4] Component [CompB] is not connected with the following
    methods provided essential for its operation: [Metz]
  - Port [P4] Component [CompB] is not connected with the following
    methods requireds waiting for connection: [metW]
- Cycles of the service execution
  Not identified.
--------------------------------------------------------------------
Behavioral Analysis with Errors
--------------------------------------------------------------------
```

Figure 9. Behavioral Analysis Report with error.

The developed tools automate the proposed analysis approach and the tests have shown the ability to automatically locate structural and behavioral errors.

As future work, we highlight the need of assessing the produced automatic support in the development of larger applications, the development of automated support to assist the creation of component adapters and to find alternatives to assess functional compatibility. Thus, we expect the possibility of producing component-oriented software specification more accurately, less prone to error, and improve its quality.

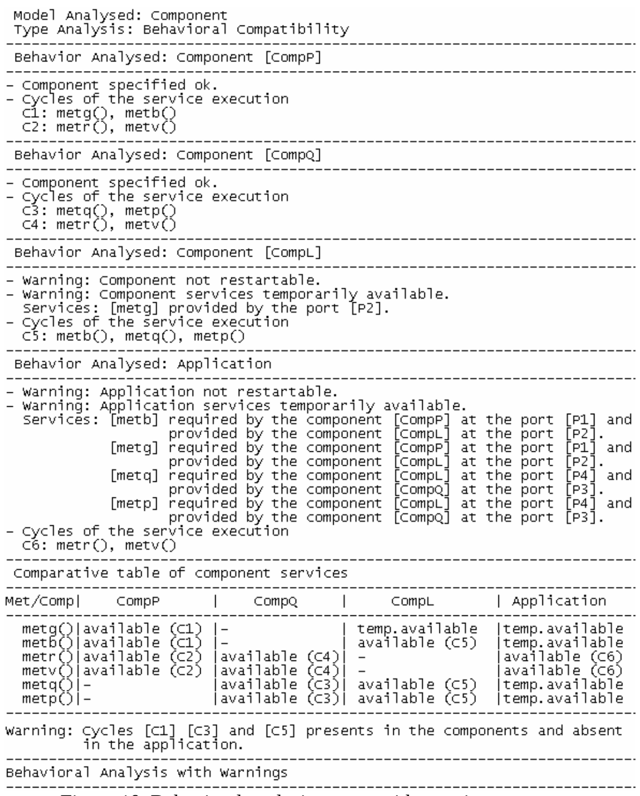
REFERENCES

[1] Object Management Group. Unified Modeling Language: Superstructure version 2.4. Available in: <http://www.omg.org/spec/UML/2.4/Superstructure/Beta2/PDF>. Access: 20 January 2011.

[2] C. Szyperski, Component Software: beyond object-oriented programming, 2.ed. Boston, EUA: Addison-Wesley Professional, 2002.

[3] M. S. Dias, and M.E.R. Vieira, "Software Architecture Analysis based on Statechart Semantics" in *Proceedings of the 10th Internacional Workshop on Software Specification and Design.* [S.1]: IEEE Computer Society, 2000.p.133.ISBN 0-7695-0884-7.

[4] SPIN. Available in: <http://spinroot.com/spin/whatisspin.html>. Access: 10 November 2010.



```
 Model Analysed: Component
 Type Analysis: Behavioral Compatibility
--------------------------------------------------------------------------------
 Behavior Analysed: Component [CompP]
--------------------------------------------------------------------------------
- Component specified ok.
- Cycles of the service execution
  C1: metg(), metb()
  C2: metr(), metv()
--------------------------------------------------------------------------------
 Behavior Analysed: Component [CompQ]
--------------------------------------------------------------------------------
- Component specified ok.
- Cycles of the service execution
  C3: metq(), metp()
  C4: metr(), metv()
--------------------------------------------------------------------------------
 Behavior Analysed: Component [CompL]
--------------------------------------------------------------------------------
- Warning: Component not restartable.
- Warning: Component services temporarily available.
  Services: [metg] provided by the port [P2].
- Cycles of the service execution
  C5: metb(), metq(), metp()
--------------------------------------------------------------------------------
 Behavior Analysed: Application
--------------------------------------------------------------------------------
- Warning: Application not restartable.
- Warning: Application services temporarily available.
  Services: [metb] required by the component [CompP] at the port [P1] and
                   provided by the component [CompL] at the port [P2].
            [metg] required by the component [CompP] at the port [P1] and
                   provided by the component [CompL] at the port [P2].
            [metq] required by the component [CompL] at the port [P4] and
                   provided by the component [CompQ] at the port [P3].
            [metp] required by the component [CompL] at the port [P4] and
                   provided by the component [CompQ] at the port [P3].
- Cycles of the service execution
  C6: metr(), metv()
--------------------------------------------------------------------------------
 Comparative table of component services
--------------------------------------------------------------------------------
Met/Comp|    CompP       |    CompQ      |     CompL        | Application
--------------------------------------------------------------------------------
  metg()|available (C1) |-              | temp.available   |temp.available
  metb()|available (C1) |-              | available (C5)   |temp.available
  metr()|available (C2) |available (C4)| -                |available (C6)
  metv()|available (C2) |available (C4)| -                |available (C6)
  metq()|-              |available (C3)| available (C5)   |temp.available
  metp()|-              |available (C3)| available (C5)   |temp.available
--------------------------------------------------------------------------------
Warning: Cycles [C1] [C3] and [C5] presents in the components and absent
         in the application.
--------------------------------------------------------------------------------
Behavioral Analysis with Warnings
--------------------------------------------------------------------------------
```

Figure 10. Behavioral analysis report with warnings.

[5] PROMELA. Process or Protocol Meta Language. Available in: <http://www.dai-arc.polito.it/dai-arc/manual/tools/jcat/main/node168.html>. Access: 10 November 2010.

[6] S. Chouali, M. Heisel, and J. Souquières, "Proving Component Interoperability with B Refinement," in *International Workshop on Formal Aspect on Component Software*, H. R. Arabnia and H.Reza, Eds. CSREA Press, 2005, to appear in ENCTS 2006.

[7] Atelier-B. Available in: <http://www.atelierb.eu/index-en.php>. Access: 10 November 2010.

[8] I. Mouakher, A. Lanoix, and J. Souquières, "Component Adaptation: Specification and Verification," in *Proceedings of the International Workshop on Component-Oriented programming, (WCOP).* 2006

[9] A. Braccialia, A. Brogi, and C. Canal, "A formal approach to component adaptation", in *Journal of Systems and Software* Volume 74, Issue 1, 1 January 2005, Pages 45-54.

[10] R. P e Silva, "Suporte ao Desenvolvimento e Uso de Frameworks e Componentes," PhD Dissertation , Porto Alegre, UFRGS/II/PPGC, march 2000.

[11] R. P e Silva, Como Modelar com UML 2, Florianópolis: Visual Books, 2009. ISBN: 978-85-7502-243-6.

[12] A. Coelho, "Reengenharia do Framework OCEAN," M.Sc. Thesis, Florianópolis, UFSC. 2007.

[13] T. C. de S. Vargas, "Suporte à Edição de UML 2 no Ambiente SEA," M.Sc. Thesis, Florianópolis, UFSC. 2008.

[14] J. A. Saldhana and S. M. Shatz, "UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis," in *International Conference on Software Engineering and Knowledge Engineering*. Proc. of the Int. Conf. On Software Eng. and Knowledge Eng. (SEKE), Chicago, 2000.

[15] Pipe. Platform Independent Petri net Editor 2, versão 2.5. Available in: <http://pipe2.sourceforge.net/>. Access: 20 January 2011.