International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 4 (2012) pp. 641-647


# Automatic Criteria-based Configuration for the Component Selection Problem

**Andreea Vescan and Horia F. Pop**

Babeş-Bolyai University, Faculty of Mathematics and Computer Science,
1 M. Kogalniceanu, Cluj-Napoca 400084, Romania
{*avescan, hfpop*}*@cs.ubbcluj.ro*

***Abstract***: **Component Based Software Engineering (CBSE) is concerned with the assembly of pre–existing software components that leads to a software system that responds to client–specific requirements. Component selection and component systems assembly have become two of the key issues involved in this process.**
**This work presents an approach for component selection, solution based on the ratio of number of satisfied requirements to the cost of the component as a measure to maximize our heuristic decision. This paper also addresses the problem of automatic assembly of component systems. The algorithm *CSACC* (*Component System Automatic Criteria-based Configurations*) is presented. With our case study we show that our approach is efficient and generally applicable in practical scenarios.**
***Keywords***: Automatic configuration, Component Selection Problem, Component assembly.

## I. Introduction

Since the late 90's Component–Based Software Engineering (CBSE) is a very active area of research and development. CBSE [1] covers both component development and system development with components. There is a slight difference in the requirements and business ideas in the two cases and different approaches are necessary. Of course, when developing components, other components can be (and often must be) incorporated and the main emphasis is on reusability. Development using components is focused on the identification of reusable entities and relations between them, starting from the system requirements.
Building software applications using components significantly reduces development and maintenance costs. Because existing components can often be reused to build new applications, it is less expensive to finance their development.
In this paper, we address the problem of automatic component selection. In general, there may be different alternative components that can be selected, each coming with their own set of offered requirements and cost. We aim at a selection approach that guarantees the optimality of the generated component-based system, an approach that considers at each step the component with the maximum ratio of offered functionalities to the cost of the component. The compatibility of components is not discussed here, it will be treated in a future development.
We discuss the proposed approach as follows. Related work on *Component Selection Problem* is discussed in Section II. Our approach for the *Component Selection Problem*: the formal statement of the Component Selection Problem, the elements/notations used for composing the system and the new algorithm *CSACC* (*Component System Automatic Criteria-based Configurations*) is introduced in Section III. The result of the algorithm may be a valid solution or a partial solution when the set of available components is not sufficient to obtain a solution. An error code is used to specify the type of obtained result. Using an example we discuss the new selection decision. We conclude our paper and discuss future work in Section V.

## II. Related work

A framework for automating component retrieval and adaptation for software reuse is described in [2]. They used layered architecture using feature–based, signature–based and specification-based retrieval engines to retrieve components that completely or partially match a problem.
MaDcAr [3] provides an uniform solution for automating both the construction of applications from scratch and the adaptation of existing component assemblies. A MaDcAr compliant engine computes a configuration and builds the application, and when the execution context changes, it chooses a more appropriate configuration and re-assembles the components accordingly.
An algorithm for selecting COTS components with multiple interfaces from a repository in order to implement a given software architecture is presented in [4]. The traditional interface operators are extended to the case in which components supports more than one interface.
The unified approach to the construction of component systems by employing methods from the area of compiler construction and especially optimizing code was proposed in [5]. The approach allows to first select an optimal set of components and adapters and afterwords to create a working system by providing the necessary glue code.
In [15] the authors argue that ideally component models should include both design and deployment phases, and it should be possible to compose components in both phases. They also demonstrate a preliminary implementation of com-

position in both phases in a component model we have defined.
The preliminary ideas for a proposal of a component selection method whose development is fast, agile and requirements driven is presented in [16]. The presented method is called COTSRE and it is based on SIREN, which is a method of RE based on the standards of this discipline and the use of reuse requirements catalogs. The final aim of the proposal is to achieve an RE method that guides the selection, the development and the composition of a set of independent software components, and whose application is the simplest possible.
In [6] constructing all possible components configurations using parallel and serial composition was proposed. The *APCC* algorithm (*All Possible Components Configurations*) is based on the idea of composition; components are used as building blocks from a repository and assembled or plugged together into larger blocks or systems. The main steps of *APCC* algorithm are: check disjoint in/out conditions; check if composition is possible; compute the interdependencies and compute all possible components configurations based on interdependencies.
The execution elements for the component–based model proposed in [6] is presented in [7]: the possible operations (propagation and evaluation) and the state of execution. The construction of a component-based model does not need much time and effort. Just wire the component's output port with other component's input port and the ports of the final component system with the ports of the desired component.
The execution of the model starts with the initialization, which consist of initialization of the input port of the first component. Execution starts after we have chosen an assembly from the set of available (generated) configurations. If at a given time, both types of operation can be performed, the propagation operation is chosen. Between many evaluation operations, one component is chosen randomly.
The previous approach was improved by obtaining a solution with a minimum number of components in [14]. The selection process considered the component with the maximum number of provided operations. In this current paper we improve the selection process by considering different criteria like: the dependencies, the cost of the component and the ratio of the number of provided operations to the cost.

## III. Constructing component-based system by automatic component selection

In Component–Based Software Engineering the construction of cost-optimal component systems is a nontrivial task. It requires not only to optimally select components but also to take their interplay into account.
We assume the following situation: given a repository of components and a specification of the component system that we want to construct (set of final requirements), we need to choose components and to connect them such that the target component system fulfills the specification. The following information must be specified: the description of the required services (set of requirements) and the component specifications, which consist of the provided services and required services, as well as the dependencies (contexts).
Informally, our problem is to select a set of components from an available component set which can satisfy a given set of requirements while minimizing number of selected components and the cost. Each component has assigned a set of requirements it satisfies.
**Problem statement.** Given a set of specified components and a set of required operations we want to automatically obtain a component assembly that offers the desired functionalities (by selecting at each step the component that provides a maximum number of required operations) using a minimum set of components with a minimum cost.

### *A. Formal Statement of the Component Selection Problem*

Component Selection Problem (CSP) is the problem of choosing the minimum number of components from a set of components such that their composition satisfies a set of objectives (a variation of CSP, the cost of each component is not considered). The notation used for formally defining CS, as laid out in [8] with a few minor changes to improve appearance is described in what follows.
Denote by $SR$ the set of final system requirements (target requirements) $SR = \{r_1, r_2, ..., r_n\}$, and by $SC$ the set of components available for selection $SC = \{c_1, c_2, ..., c_m\}$. Each component $c_i$ can satisfy a subset of the requirements from $SR$, $SR_{c_i} = \{r_{i_1}, r_{i_2}, ..., r_{i_k}\}$.
The goal is to find a set of components $Sol$ in such a way that every requirement $r_j$ from the set $SR$ may have assigned a component $c_i$ from $Sol$ where $r_j$ is in $SR_{c_i}$, while minimizing the number of components in the solution $Sol$.

### *B. Elements for composing the system*

**Component specification.** One of the most popular definitions of a component was offered by a working group at ECOOP (European Conference on Object-Oriented Programming).
**Definition 1** *([9]) A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and it is subject to composition by third parties.*
Although all definitions of component differ in detail, they assert that a component is an independent software package that provides functionality via well-defined interfaces. Moreover, they all emphasize the importance of well–defined interfaces. The interface could be an export interface through which a component provides functionality to other components or an import interface through which a component gains services from other components. They also emphasize the "black-box" nature of a component: that is, a component can be incorporated in a software system without regard to how it is implemented.
The component in its simplest form contains code that can be executed on some platform and an interface that provides the access to the component. A component is considered to be a black box, i.e., its internals inaccessible to the component user. Hence, interfaces are the only access points to the component and the specification of the component comes down to the specification of the component interfaces.
Specification of the component interfaces in the current component-based systems is done only on the syntactical

level. Provided interfaces are the one that contain operations that a component provides to other components or to the component user, while required interfaces are the one that contain operations used by the component. Each interface consists of a number of operations which might have input, output or input/output parameters.
Considering the above discussion we specify a component as in the following structure:

**component** $= (id,\ contexts)$,
**context** $= (provided\_Interfaces,\ required\_Interfaces)$,
**provided_Interface** $= (provided\_Operations)$ denoted by $pIOps$,
**required_Interface** $= (required\_Operations)$ denoted by $rIOps$.

In Figure 1 an example of component specification is presented. The component has two contexts with different number of required and provided interfaces: first context has one provided interface and two required interfaces and the second context has two provided interfaces and one required interface.

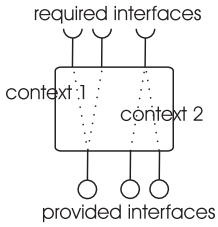


**Figure. 1**: Context component specification

Each component from the set of component $SC$ must satisfy the following condition (the set of operations of provided interfaces and the set of operations of required interfaces are disjoint - for the same context):

$$pIOps \cap rIOps = \emptyset.$$

Specialized components are also needed: components for reading the input data of the problem (source components) and components needed for displaying the output data (result) of the algorithm (destination components).
**Final system requirements specification.** The system we want to build must be specified by given required operations. Each operation is encapsulated in one required interface. The required interface is assigned to a "fictive" destination component.

### C. Criteria discussion for component selection

The decision that has to be made during the process of component selection may be based on different criteria: some cost measures based on component properties or metrics for different quality attributes or some defined component selection policy to compute automatically assembling decisions.
In a previous paper [14] the component with the maximum number of provided operations that are needed by the final system was considered at each step of the algorithm. It was discussed that the final solution may be improved if the selection criteria is upgraded with the cost measure of the component. Also the dependencies between the components involved in the composition (not only intern–dependencies for each component) could be considered. In the current paper the above stated criteria are considered.
We have developed a Greedy algorithm for Component Selection Problem where the decision selection was improved from the existing approaches ([8], [10], [11], [12]) by considering dependencies [13] between the components.

### D. Component System Automatic Criteria-based Configurations algorithm

The algorithm finds a component assembly starting from the final system requirements (operations are encapsulated in required interfaces) and selecting at each step the component with the following criteria: cost, ratio of the number of provided operations to the cost of the component, number of provided operations. The final system is obtained when there are no more required operations to be satisfied, i.e. the final components added to the solution are components specialized in reading the input data.

**Algorithm 1** CSACC algorithm (Component System Automatic Criteria-based Configurations)

**Input:**
- the number $noCompo$ of components;
- the set $valCompo$ of components;
- the number $noAllReq$ of final requirements;
- the set $valAllReq$ of final requirements.

**Output**
- the number $noRez$ of final results;
- the set $valRez$ of final results;
- the $errorCode$ for partial solution $-1$, if exist solution error code is 1.

**Algorithm** *CSACC(noCompo,valCompo,noAllReq,valAllReq, noRez,valRez,errorCode)* is:
**Begin**
allRequiredSatisfied:=*false*;
existComponentsToConsider:=*true*;
errorCode=1;
**while** (not allRequiredSatisfied) and (existComponentsToConsider) **do**
  oneFinalRequired:=false;
  **while** (not oneFinalRequired) and (existComponentsToConsider) **do**
    *AddToSearch(noAllReq,valAllReq,noR,valR);*
    *FindProvCompo(noCompo,valCompo,noR,valR, noRez,valRez);*
    **if** (noR=0) **then**
      oneFinalRequired:=*true*;
    **else**
      existComponentsToConsider:=*false*;
    **endif**
  **endwhile**
  **if** (noAllReq=0) **then**
    allRequiredSatisfied:=*true*;
  **endif**
**endwhile**
**if** (existComponentsToConsider=*false*) **then**
  errorCode:=-1;
**endif**
**End.**

As algorithm prerequisites we must first check if it is possible to obtain a solution:

- Each required operation (from the final system specification) must be found as a provided operation of one of the components from the repository.
- Each required operation (from each required interface) must have a provided operation in one of the components from the repository.

The result of the algorithm may be a valid solution or a partial solution (when the set of available components is not sufficient to obtain a solution). An error code is used to specify the type of obtained result. The best solution is computed, i.e. the one with the minimum number of used components and with the minimum cost.
The subalgorithm *AddToSearch* adds the current requirements from the given set of final system requirements ($noAllReq$, $valAllReq$) to a list of current requirements that are need next to be satisfied ($noR$, $valR$).
In the proposal [14] the subalgorithm *FindProvCompo* finds the component (from the set of given components $noCompo$, $valCompo$) with maximum number of provided operations that satisfy the current requirements, adds them to the set of final results ($noRez$, $valRez$) and updates the set of current requirements with the set of required operations of the selected component.
In the current approach the subalgorithm *FindProvCompo* selects the next component taking into considerations the following criteria:

- the ratio of number of requirements satisfied to the cost of the component as a measure to maximize our heuristic decision:

$$|SRc_i \bigcap RSR|/cost(c_i)\ is\ maximal,$$

- the number of dependencies of the considered $c_i$ component $is\ minimal$.

**Example**
In order to best emphasize our approach we use the following example.
In the repository there are seven components that are specified according to Section III-B. We describe here (see Table 1 for details) only the specification for the first component (the specification of the other components is similar). The final system requirements are specified and each required operation is encapsulated in one required interface. The solution described in Figure 2 is obtained running the Algorithm 1.

*Table 1*: Component $C_1$ Contexts Specification

| $Component_1$ | ProI | ProOp | ReqI | ReqOp |
|---|---|---|---|---|
| $Context_1$ | 1 | $P\_op1$ | 1 | $R\_op11$ |
| | | | 2 | $R\_op12$ |
| $Context_2$ | 2 | $P\_op3$ | 3 | $R\_op13$ |

Note that in Figure 2 the operations that are "compatible" have the same name, except the difference between provided and required type: for example, the fifth component has a required operation $R\_op32$ and the seventh component has a provided operation $P\_op32$. The compatibility of components is not discussed here, it will be treated in a future development.
The input of the algorithm are the set of specified components and the set of required operations of the final system. Each required operations is encapsulated in one required interface.
For each required interface the following steps are performed: find the component (or components) that provides the required operation(s) and then update the required interface by adding the required operations of the selected component; for the new required interface (for each required operations) find again the component (or components) that offers a maximum provided operations for the required ones and with the minimum cost, and then update the required interface; the process continues until there are no required operations to be satisfied or no solution is found (no component(s) can provide operations for the unsatisfied remained required operations).

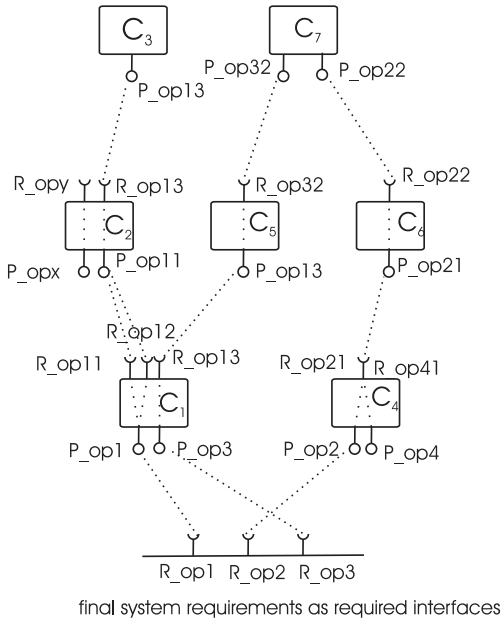


**Figure. 2**: Obtained solution of the considered example after applying the Component System Automatic Criteria-based Configurations (CSACC) algorithm

The final system requirements are satisfied by all the components from the set of components. For the given requirements of the final system a solution is found (see Figure 2).

## IV. Experiments and comparisons

A short and representative case study is presented in this section. Starting for a set of six requirements and having a set of ten available components and the dependencies between the requirements of the components, the goal is to find a subset of the given components such that all the requirements are satisfied, a system with a minimum cost.
The set of requirements $SR = \{r_0, r_1, r_2, r_3, r_4, r_5\}$ and the set of components $SC = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9\}$ are given. In Table 2 the cost of each component from the set

of components $SC$ is presented.

*Table 2*: Cost values for each component in the $SC$

| Comp | $c_0$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ | $c_6$ | $c_7$ | $c_8$ | $c_9$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Cost | 8 | 7 | 6 | 9 | 6 | 14 | 15 | 14 | 7 | 14 |

Table 3 contains for each component the provided services (in terms of requirements of the final system) and Table 4 the dependencies between each requirement from the set of requirements.

*Table 3*: Requirements elements of the components in $SC$

| | $c_0$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ | $c_6$ | $c_7$ | $c_8$ | $c_9$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $r_0$ | √ | | √ | √ | | | | | | √ |
| $r_1$ | | | | | √ | | | | √ | |
| $r_2$ | | √ | | | | √ | | | √ | |
| $r_3$ | √ | | | | | | √ | | | |
| $r_4$ | | | | | | √ | √ | √ | | √ |
| $r_5$ | | √ | | | | | √ | √ | | √ |

*Table 4*: Specification Table of the Requirements Dependencies

| Dependencies | $r_0$ | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ |
|---|---|---|---|---|---|---|
| $r_0$ | | √ | | | | |
| $r_1$ | | | | | | |
| $r_2$ | √ | | | | √ | |
| $r_3$ | | √ | | | | |
| $r_4$ | √ | | | | | |
| $r_5$ | | √ | | | | |

### A. Results obtained by the previous approach

In the current section we discuss the application of the CSAC [14] algorithm. For the set of available components there are two components that may provide the needed operations and have no dependencies: $c_4$ offers $r_1$ and $c_8$ offers the same requirement $r_1$ but may also provide $r_2$ if $\{r_0, r_5\}$ will be next provided. Using the maximal criteria of the CSAC [14] algorithm, the $c_8$ component is selected.
The remained set of requirements is: $\{r_0, r_2, r_3, r_4, r_4\}$. The next possible selected components that offer two requirements are: $c_0$ offers $\{r_0, r_3\}$, component $c_6$ offers $\{r_3, r_5\}$ and component $c_9$ offers $\{r_0, r_5\}$. But, for component $c_9$ by now providing $\{r_0, r_5\}$, the $c_8$ component may provide also $r_2$, and $r_4$ is provided by $c_9$ (possible because is another used context). The $c_9$ component is selected.
Only one requirement must be satisfied: $r_3$. There are three possible components that may provide it: $\{c_5, c_6, c_7\}$. The cost of the founded solution for each possible selection is: 35, 36 and 35.

### B. Results obtained by the current improved approach

In the current section we discuss the application of the CSACC algorithm considering the ratio of number of requirements satisfied to the cost of the component as a measure to maximize our heuristic decision, and the dependencies between the components.
For the given requirements of the final system, the found solution is presented in Figure 3.
For the set of available components there are two components that may provide the needed operations and have no dependencies: $c_4$ offers $r_1$ and $c_8$ offers the same requirement $r_1$ but may also provide $r_2$ if $\{r_0, r_5\}$ will be next provided. The ratio are: $1/6$ and $2/7$, thus the component $c_8$ being selected.
The next set of components that may provide required functionalities are: $c_0$ with the ratio $2/8$, component $c_2$ with the ratio $1/6$ and the component $c_9$ with the ratio $3/14$. The component with the maximum ratio value is $c_0$, thus the next selected component, providing requirements $\{r_0, r_3\}$.
The set of remained requirements is: $\{r_2, r_4, r_5\}$. The components that may provide one of two of the requirements are: $\{c_6, c_7, c_9\}$. The component with the maximum ratio is $c_7$ (or $c_9$). The obtained solution contains the components: $\{c_8, c_0, c_7\}$ and the cost is 29.

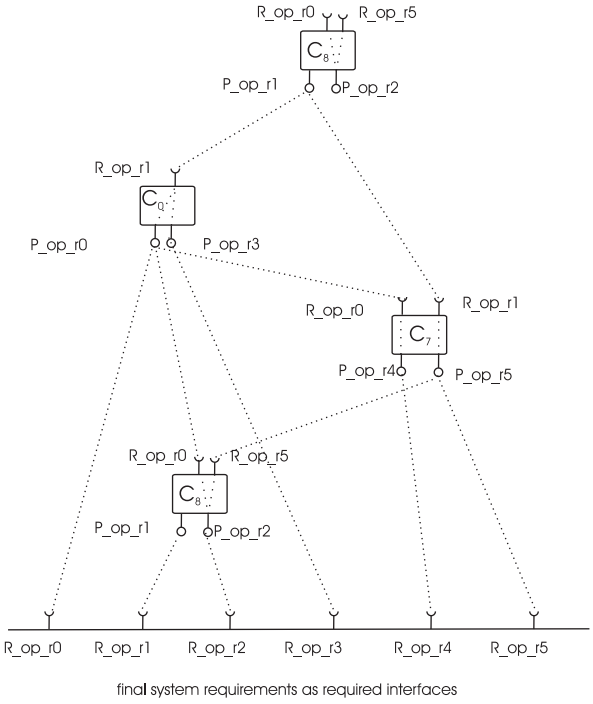


**Figure. 3**: Obtained solution of the considered case study after applying the Component System Automatic Criteria-based Configurations (CSACC) algorithm

### C. Discussion

The two approaches find different solutions with different final cost. Although the same solution could be found (for a proper instance of the given set of requirements, components and component costs and dependencies) the previous approach finds the solution with a higher cost.

## V. Conclusion and future work

CBSE is the emerging discipline of the development of software components and the development of systems incorporating such components. A challenge in component-based software development is how to assemble components effectively and efficiently.
This paper presents the following main contributions: a proposal for component selection (the component with the maxi-

mum ratio of the number of satisfied requirements to the cost of the component) and an algorithm named *CSACC* (*Component System Automatic Criteria-based Configurations*) for automatic assembly of component systems. A real world system application will be considered next to validate our approach.
We intend to extend our approach by specifying and proving the compatibility between two connected components. The protocol for each provided operations of a component have to be specified and included into the composition process.

## Acknowledgments

This material is based upon work supported by the Romanian National University Research Council under award PN-II no. ID_550/2007.

## References

[1] I. Crnkovic, M. Larsson. Building Reliable Component-Based Software Systems, Artech House publisher, 2002.

[2] B. Morel. SPARTACAS Automating Component Reuse and Adaptation, IEEE Transactions on Software Engineering, 30 (9), pp. 587-600, 2004.

[3] G. Grondin, N. Bouraqadi, L. Vercoute, A. Andrews. MaDcAr: An Abstract Model for Dynamic and Automatic (Re-)Assembling of Component-Based Applications, In Proceedings of the $9_{th}$ International SIGSOFT Symposium on Component-Based Software Engineering, pp. 360-367, 2006.

[4] L. Iribarne, J. M. Troya, A. Vallecillo, A. Andrews. Selecting Software Components with Multiple Interfaces, In Proceedings of the $28_{th}$ Euromicro Conference, pp. 1089-6503, 2002.

[5] L. Gesellensetter, S. Glesner. Only the Best Can Make It: Optimal Component Selection, Electronic Notes in Theoretical Computer Science, 176 (2), pp. 105-124, 2007.

[6] A. Fanea, S. Motogna. A formal model for component composition, In Symposium "Zilele Academice Clujene" (Babes-Bolyai University, Cluj-Napoca, Romania, June 14-22 2004), pp. 160-167, 2004.

[7] A. Fanea. Specification, Construction and Execution of a Component-Based Model, In Symposium "Colocviul Academic Clujean de Informatica" (Babes-Bolyai University, Cluj-Napoca, Romania, June 1-2 2005), pp. 87-92, 2005.

[8] M. R. Fox, D. G. C. Brogan, P. F. Reynolds. Approximating component selection, In Proceedings of the $36_{th}$ conference on Winter simulation, pp. 429-434, 2004.

[9] C. Szyperski. Component Software: Beyond Object-Oriented Programming, Addison-Wesley Professional, 1997.

[10] N. Haghpanah, S. Moaven, J. Habibi, M. Kargar, S. Hassas Yeganeh. Approximation Algorithms for Software Component Selection Problem, In Proceedings of the $14_{th}$ Asia-Pacific Software Engineering Conference, pp. 159-166, 2007.

[11] C. Alves, J. Castro. Cre: A systematic method for cots component selection, In Proceedings of the Brazilian Symposium on Software Engineering, pp. 195- 207, 2001.

[12] P. Baker, M. Harman, K. Steinhofel, A. Skaliotis. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem, In Proceedings of the $22_{nd}$ IEEE International Conference on Software Maintenance, pp. 176-185, 2006.

[13] A. Vescan. Dependencies in the component selection problem, Creative Mathematics and Informatics, 17 (3), pp. 532-537, 2008.

[14] A. Vescan, H. F. Pop. Automatic configuration for the component selection problem, In Proceedings of the $5_{th}$ International Conference on Soft computing as transdisciplinary sicence and technology, pp. 479-483, 2008.

[15] K.-K. Lau, L. Ling, P. V. Elizondo. Towards Composing Software Components in Both Design and Deployment Phases, In Proceedings of the $10_{th}$ International SIGSOFT Symposium on Component-Based Software Engineering, pp. 274-282, 2007.

[16] M. A. Martinez, A. Toval. COTSRE: a COmponenTs Selection method based on Requirements Engineering, In Proceedings of the Seventh International Conference on Composition-Based Software Systems, pp. 220-223, 2008.

## Author Biographies

**Andreea VESCAN** graduated the B.Sc. program in Computer Science at the Faculty of Mathematics and Computer Science in 2003. After graduation, Andreea was enrolled in a master degree programme in Computer Science within the topic Component-Based Programming, in the academic year 2003/2004. During her master studies she was also a computer science teacher at the Informatics Highschool Tiberiu Popoviciu, Cluj-Napoca. She became a PhD student in 2004, developing an active research work, both theoretical and applicative, being able to work either in a team or alone. In October 2008 she obtained her Ph.D. degree in Computer Science at Babes-Bolyai University, with the thesis "Construction Approaches for Component-Based Systems". Since 2009 she works as a Lecturer Professor of Computer Science at Babes-Bolyai University.

She was a beneficiary of many international mobility programmes, including a CEEPUS mobility (Paisii Hilendarski University, Plovdiv, Bulgaria, August 2003 and Eotvos Lorand University, Budapest, Hungary,

June 2006) and the international Marktoberdorf Summer School in Germany, Software System Reliability and Security, still in 2006. In the summer of 2008 she was a research scholar at the Wayne State University Detroit, USA.

Mrs. Vescan participated in many research grants in area related to software components, methods for data analysis, scientific computing and optimization for interdisciplinary applications. The research and teaching interests of Mrs. Vescan, are primarily in formal models for component-based systems and verification and validation of software systems. She has published, alone or in co-operation, around 40 scientific papers, among which around 10 are indexed in ISI Web of Knowledge.

**Horia F. POP** graduated in Computer Science from the Babes-Bolyai University of Cluj-Napoca (UBB) in 1991 and since October 1991 is with the Department of Computer Science, Faculty of Mathematics and Computer Science, UBB. Since February 2004 he works there as Professor of Computer Science. In November 1995 he obtained his Ph.D. degree in Computer Science at UBB, with the thesis “Intelligent Systems in Classification Problems”. He is teaching courses, seminars and practical works in areas related to Artificial Intelligence. He has published, alone or in co-operation, around 100 scientific papers, among which around 30 are indexed in ISI Web of Knowledge. He developed scientific visits at different western universities and institutions, among which we mention here the development of an NSF contract on Soft Computing with the University of Memphis (2000-2003), the participation to a DFG program on Natural Language Processing with the University of Hamburg (1999-2001), and the visiting professorship at the Wayne State University in Detroit, US (2006). His research interests are Fuzzy Sets, Intelligent Data Analysis, Soft Computing, and Artificial Intelligence.