Babeş–Bolyai University
Faculty of Mathematics and Computer Science
1, M. Kogălniceanu Street
400084, Cluj–Napoca, România
http://www.ubbcluj.ro

Construction Approaches for Component–Based Systems

PhD Thesis Abstract

PhD Student: Andreea (Fanea) Vescan
Advisor: Prof. Dr. Militon Frenţiu

2008

DEDICATION

To

My parents

and

In loving memory of my grandparents!

To

My husband Cristi.
None of this would be possible without your love and support!

To

My teachers!

ACKNOWLEDGMENTS

This dissertation would have never been finished without the help of many people, to whom I would like to express my sincere gratitude.

First of all, I want to thank my adviser Prof. Militon Frenţiu for his patience and encouragement. He trusted my ability to complete this work even before I had started it. In particular, I really think that he is one of the most "beautiful minds" in the areas of Software Engineering and Computer Science. Moreover, although he is born to be a leader, he really appreciates all people that collaborate with him. He really has a great heart beyond being a very smart person.

I dedicate this dissertation to my family. I would like to warmly thank my parents, Cornelia and Voicu, and my brother Dan, for all their love, for all their mental and financial support and for believing in me. In particular I am very thankful to my mom for the answers to all the profound questions of my childhood and to my dad for teaching me to approach all things systematically.

Very special thanks to my husband Cristi for giving me confidence during this research, for the unique way in which he loves and understands me day by day, for his infinite patience in supporting me during each moment of my PhD programme. His moral support, patience, and humor have made everything possible.

This thesis is all dedicated to you! Thanks all, I am really lucky to have a family like this.

Finally, let me thank the Computer Science Department staff and all my colleagues. Many thanks go to Prof. Bazil Pârv for valuable suggestions and stimulating discussions. I thank Prof. Horia Pop for accepting me in his research group and for his guidance, enthusiasm and support. Many thanks also to all of the colleagues who have helped me and collaborated with me (Simona Motogna, Laura Dioşan, Crina Groşan, Camelia Şerban, Camelia Pintea) during my work on the PhD thesis.

Contents

Introduction

This PhD thesis is the result of my research in Software Engineering, particular in the domain of Component–Based Software Engineering (CBSE), started in 2004.

Since the late 90's CBSE is a very active area of research and development. CBSE [CL02] 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.

There are two issues [CL02] needed to be addressed when a software system is to be constructed from a collection of components:

- *Component integration* – the mechanical process of wiring components together. There has to be a way to connect the components together.
- *(Behavior) Component composition* – we have to get the components to do what we want. We need to ensure that the assembled system does what is required. The constituent components must not only plug together, they must perform well together.

Building software applications using components significantly reduces software complexity, increasing software reuse. In addition, it reduces the development time, efforts and costs, resulting in a better software quality.

The goal of this thesis is to investigate automated techniques that may be used to support component structural composition (integration). Our problem is to select a number of components from an available set such that their composition satisfies a set of objectives (the final system requirements). The *Simple Component Selection Problem* computes all the possible component configurations and the *Multicriteria Component Selection Problem* finds the best component assembly satifying some criteria.

This thesis focuses on the activity of component selection and integration. It contains 107 bibliographical references and is divided in nine chapters as follows.

First chapter provides an introduction to the field of Component–Based Software Engineering and general issues about selection, integration and composition of components are stated. The used approaches for component–based systems construction are described in details: backtraking–based approaches, automata–based approaches, evolutionary–based approaches, Greedy approach, and Branch and Bound approach.

Chapter 2, **Backtracking–based composition approach**, introduces a new approach for construction of component configurations based on the backtracking method. Two component configurations, one based on data dependencies (***APPC*** – ***A****ll* ***P****ossible* ***C****omponent* ***C****onfigurations* algorithm) and the other based on temporal dependencies (***TCCR*** – ***T****emporal* ***C****omponent* ***C****omposition* ***R****estraint* algorithm) are proposed. The chapter also describes a component assembly execution model. New rules for execution extends the previously defined execution model.

Chapter 3, **Automata–based composition approach**,describes (using an automata representation of a component–based system) two introduced approaches for component configurations construction: one is based on input data dependencies (*MakeAllModels* algorithm) and the other is based on task dependencies (***C****ontrol****F****low and* ***D****ata****F****low* ***S****yntactic* ***C****omposition* algorithm).

Chapter 4, **Artificial intelligence–based composition approach**, presents different intelligent–based approaches to component configurations: using Genetic Algorithms – an approach to analyze component integration and another approach to analyze behavioral component composition using Cartesian Genetic Programming, and using Genetic–based Programming – an approach to determine optimal component combinations using Multi Expression Programming and another approach to design a component–based system using Membrane Computing (P–Systems).

Chapter 5, **Multicriteria Component Selection Problem**, presents Greedy, Branch and Bound and Evolutionary approaches for the *Multicriteria Component Selection Problem*, using various criteria. The Greedy selection function (to best select the component to be added to the final solution) uses a ratio of component cost and requirements. The dependencies are also considered. The Branch and Bound approach considers at each step the cost of the selected component and the set of remaind requirements to be satisfied. The dependencies are used during the selection of the successors. Two evolutionary representations are used with two ways to deal with the multiobjective optimization problem, i.e. weighted sum method and Pareto dominance principle.

Chapter 6, **Component Adaptation Architectures. A Formal Approach**, provides a formal mathematical model for component function adaptation. Four component adaptation architectures are presented. The behavior adaptation constraints for each architecture (Serial or Sequential Adaptation Architecture – SAA, Parallel Adaptation Architecture – PAA, Alternative Adaptation Architecture – AAA and Repetitive Adaptation Architecture – RAA) are discussed.

Chapter 7, **Metrics in Component–Based Software Engineering**, defines a set of new metrics to quantify quality attributes of a component–based system and a new set of metrics to best select a component assembly from a set of available configurations.

Chapter 8, **Component Backtracking and Ant Colony–based approach to solve TSP, Labyrinth Problem and AGAP**, presents the developed execution model to solve Traveling Salesman Problem using backtracking algorithms and Ant (Colony) Systems. The chapter also describes the applicability of the developed execution model to solve the Labyrinth Problem and the Airport Gate Assignment Problem using Ant Colony Systems.

Chapter 9, **Conclusions and future work**, draws the main conclusions about our approaches and several potential improvements of our work.

The original contributions introduced by this thesis are contained in Chapters 2, 3, 4, 5, 6, 7, 8 and they are as follows:

- A new approach for the construction of component configurations [FM04] (Subsection 2.1.2) based on the backtracking method.
- A temporal component restraint configuration [Ves06, Ves07a] (Subsection 2.1.3) based on the backtracking method.
- Execution model of a component configuration [Fan05] (Subsection 2.2.3).
- An extension of the previously defined execution model with new execution rules [Ves07b] (Subsection 2.2.4).
- A new approach for the construction of component configurations having automata representation, proposal based on input data dependencies [FMD06] (Section 3.1).
- A new approach for the construction of component configurations having automata representation, approach based on task dependencies [VM06b, VM06a] (Section 3.2).

- A new approach to generate component execution orders based on Genetic Algorithms (GA) [FD05a] (Subsection 4.1.1). A comparison between the previously developed backtracking–based algorithm (Subsection 2.1.2) and the GA algorithm is presented.
- A new approach to analyze component integration based on Genetic Algorithms (GA) [FD06b] (Subsection 4.1.2). A mapping from the previously developed automata–based model (Subsection 2.1.2) to a GA approach is considered.
- A new approach to develop optimal component combinations using Multi Expression Programming [FD05b] (Subsection 4.2.1).
- A new approach to behavioral component composition using Cartesian Genetic Programming [FD06c] (Subsection 4.2.2).
- A new approach that uses Membrane Computing (P–Systems) to design a component–based system [FD06a] (Subsection 4.3.3).
- New approaches for the Component Selection Problem modeled as a multiple objective optimization problem [VG08a, VG08b, Ves08e, Ves08c, VGP08, Ves08d]. Different representations are used: requirements–based representation (Subsection 5.3.1) and components–based representation (Subsection 5.3.2).
- New approaches for the Component Selection Problem modeled as a multiple objective optimization problem [Ves08b, VP08] using a Greedy approach (Subsection 5.1) and a Branch and Bound approach (Subsection 5.2).
- A formal mathematical model for component function adaptation [Ves08a] (Subsection 6.2). Four component function architectures are proposed.
- A set of new metrics to quantify quality attributes of component–based systems [SV07b] (Subsection 7.1.2).
- A new set of metrics to best select a component assembly from a set of available configurations [SV07a] (Subsection 7.2.1).
- Applying the developed execution model to solve Traveling Salesman Problem using backtracking algorithms [FP05, VP06a, VP06b] (Subsection 8.1.2) and Ant (Colony) Systems algorithms [VP07] (Subsection 8.1.3).
- Applying the developed execution model to solve Labyrinth Problem (Subsection 8.2.1) and Airport Gate Assignment Problem [PV07b, PV07a] (Subsection 8.2.2) using Ant Colony Systems.

1 Setting the context

This chapter provides an introduction to the field of Component–Based Software Engineering (CBSE), given a particular emphasis to the construction of systems out of components.

1.1 Component–Based Software Engineering. Component definitions

CBSE is based on the concept of component. Components are at the heart of CBSE. We may find several definitions of a component in literature, most of which fail to give an intuitive definition of a component, but focus instead on the general aspects of a component. In what follows by a component we understand a software component.

One of the most popular definitions of a component [Szy98] was offered by a working group at ECOOP (the European Conference on Object–Oriented Programming). The definition emphasizes component composition. As a unit of composition, each component has its specified interface that determines how it can be composed with other components.

Definition 1.1.1 ([Szy98]) *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.*

Component integration and (behavior) component compositions are the two issues needed to be addressed when constructing component configurations. We need to consider how components can be combined with each other in order to build a more complex system or component. Generally, one constraint on combining components is that these components are viewed as black boxes, i.e. it is only necessary to consider their export and import interfaces, resulting that components are not allowed to be modified.

1.2 Constructing systems out of components

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. In what follows work related to constructing systems out of components is presented.

A framework for automating component retrieval and adaptation for software reuse is described in [Mor04]. Layered architecture using feature–based, signature–based and specification–based retrieval engines to retrieve components that completely or partially match a problem are used.

MaDcAr [GBV06] provides a 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 [ITV02].

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 [GG07]. 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.

Two different types of situations are possible when constructing systems out of components: obtain a *complete solution* (no extra components are needed to perform the desired tasks), or obtain a *partial solution* (extra components or assemblies are needed to complete the required tasks). Both situations are discussed in this thesis.

1.2.1 Complete system construction

Component selection methods [FBPFR04, BHSS06] are traditionally done in an architecture centric manner, meaning they aim to answer the question: given a description of a component needed in a system, what is the best existing alternative available in the market? Another type of component selection approach [HMH+07, GG07] is built around the relationship between requirements and components available for use.

Two different types of component systems construction are approached in this thesis: construct all the component–based system out of a set of given components that fullfils the given requirements (we denote this problem as *Simple Component Selection Problem – SCSP*), and construct only one component–based system satisfying different criteria (*Multicriteria Component Selection Problem – MCSP*). Both problems are described in details in what follows.

Simple Component Selection Problem

Informally, our problem is to select a subset of components (each of them satisfying a set of functionalities) and to connect them such that the target component system fulfills the needed requirements.

Simple Component Selection Problem (SCSP) is the problem of choosing a number of components from a set of components such that their composition satisfies a set of objectives. The notation used for formally defining *SCSP*, as laid out in [FBPFR04] (with a few minor changes to improve appearance) is described in what follows.

Denote by SR the set of the 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 may 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 (subset) 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}.

Various approaches for the *Simple Component Selection Problem* are used in this thesis: backtracking–based approaches, automata representation and evolutionary–based approaches. The notations used in this thesis for each proposed approach are presented in Section 1.3.

Multicriteria Component Selection Problem

Another variation of the *Simple Component Selection Problem* is that stated in [HMH+07]. In addition to the above description a cost c_i of a component is considered $cost(c_i)$. Different criteria for the best component selection from an available set are used in this thesis.

Multicriteria Component Selection Problem (MCSP) reduces to the *SCSP*, adding the following criteria: minimizing the $\sum_{c_i \in Sol} cost(c_i)$ and minimizing the number of components in the solution Sol.

The *Multicriteria Component Selection Problem* is investigated using Greedy, Branch and Bound, and Evolutionary approaches. The used notations are described in Section 1.4.

1.2.2 Partial system construction. Component Adaptation

In any component selection method, it is unrealistic to expect a perfect match between needed components and available components. A group of components that compose a system may have overlaps and gaps in required functionality. A process of adaptation is required.

Component adaptation [XW05] can be divided into three families. One is to adapt component signature properties, such as names, parameters. The second one is function adaptation: how to integrate different components both meeting partly the users requirement to satisfy the final requirement. The third one is behavior adaptation that is how to mediate the component behavior when behavior mismatches occur.

1.3. Simple Component Selection Problem

This section describes the approaches used to investigate the *Simple Component Selection Problem*. Backtracking method [FPS06], automata representation [PMP04, MPP04] and artificial intelligence techniques [Gol89, Sys91] are presented.

1.4. Multicriteria Component Selection Problem

Various approaches are used to solve the *Multicriteria Component Selection Problem*. We use a Greedy approach [FPS06], a Branch and Bound approach [FPS06] and Evolutionary Algorithms. Each approach and notations are described in details in the thesis.

1.5 Metrics–based component selection

CBSE is a very active area of research and development. Its goals, among others, are to consistently increase return on investment and time to market, while assuring higher quality and reliability than can be achieved through current software development [CL02].

In the area of software reengineering and reverse engineering, metrics are being used for assessing the quality and complexity of software systems, as well as getting a basic understanding and providing clues about sensitive parts of software systems.

Our approach regarding metrics concerns the selection of the best obtain solution from an available set of configurations. To analyze the quality of a component assembly some quality attributes must be considered. We introduce new metrics to help us predict the quality of the considered attributes for a component configuration.

1.6 Application of the developed models

Three problems are described in this section: Traveling Salesman Problem [BT85], Labyrinth Problem [PD07], and the Airport Gate Assignment Problem [DLRZ04, Bar05]. The problems are used to apply our developed component execution model.

2 Backtracking–based composition approach

Simple Component Selection Problem is investigated in this chapter. Data and temporal composition restraints are used when constructing systems out of components. All possible component configurations are constructed using a backtracking algorithm, taking into consideration only two types of composition operations: serial and parallel composition. Regarding execution, an execution model is proposed. Construction and execution elements are presented.

2.1 Constructing component configurations

This proposal is based on the [CS01, Hen00] approaches.

2.1.1 Definitions and notations

In order to specify the components we must first establish the entities involved in a component definition: domain **D** - a set that doesn't contain the *null* element; set of attributes **A** - an infinite fixed and arbitrary set; the attributes signify variables or fields; type of an attribute $x \in A : Type(x) \subseteq D$ represents the set of possible values for the attribute x.

Considering X a component over the set A of attributes, we use the following notations: $inport(X) \subseteq A$ represents the set of input ports (attributes) of the component X, $outports(X) \subseteq A$ represents the set of output ports (attributes) of the component X, $attributes(X) \subseteq A$ represents the set of attributes of the component X, and $inports(X) \cap outports(X) = \emptyset$.

Definition 2.1.1 ([CS01]) *A source over A is an attribute. If X is a source, attributes(X) and outports(X) are both defined to be the set consisting of the single attribute X. It has no inports and generates data provided as outports in order to be processed by other components.*

Definition 2.1.2 ([CS01]) *A destination over A is an attribute. If X is a destination, attributes(X) and inports(X) are both defined to be the set consisting of the single attribute X. It has no outports and receives data from the system as its inports and usually displays it, but it doesn't produce any output.*

Definition 2.1.3 ([CS01, FM04]) *A simple component over A is a 5-tuple X of the form*

$$(inports, outports, attributes, function, \prec_X),$$

where:

• $inports$ is a n-tuple $(in_1, ..., in_n)$ of attributes and $outports$ is a m-tuple $(out_1, ..., out_m)$ of attributes and $(out_1, ..., out_m)$ *are not* $\subset inports(SC)$;

• $function$ is an n-ary function $Type(in_1) \times Type(in_2) \times ... \times Type(in_n) \rightarrow Type(out_1) \times Type(out_2) \times ... \times Type(out_m)$.

• $attributes$ is defined to be the set of attributes consisting of the inports and the outports;

• the binary relation $\prec_X \subseteq (inports(X) \times outports(X)) \times outports(X)$.

There are two basic ways in which two components may depend on each other [MP02], [Hen00]: *parallel composition* ($A||B$, in which the operations performed on data are independent and there is not dependency between $outports(A)$ and $outports(B)$) and *a serial composition* ($A + B$, in which the B component expects some results from component A).

Definition 2.1.4 ([CS01, FM04]) *A compound component over A is a group of connected components (using serial or parallel composition), in which the output of a component may be used as input by another component from this group.*

Any compound component can be described with these two basic operations, no matter how many simple components it contains. Compound components are built from atomic

components (source, destination or simple components) using parallel and serial composition, depending on the interdependencies inside the compound component.
Definition 2.1.5 ([CS01]) *A component over A is either a source over A, a destination over A, a simple component over A, or a compound component over A.*

2.1.2 All Possible Component Configurations algorithm

The **APCC** (**A**ll **P**ossible **C**omponent **C**onfigurations) algorithm [FM04] 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 **APCC** Algorithm computes all possible component configurations. A backtracking algorithm [FPS06] was used.

In order to apply the **APCC** algorithm [FM04] the following steps must be first performed: check disjoint in/out condition ($inports(X) \cap outports(X) = \emptyset$ for each X component), check if composition is possible (the data may flow between the simple components involved in the composition: the inports of all the simple components are inports of the *black_box* or they may occur as outports of the other components), and compute the interdependencies. The interdependencies between components are computed as follows: we have to check if an inport in_i of a component appears as an outport of other component; if an inport of a component B appear as an outport out_j of a component A, ($in_i = out_j$) then we have to use component A before component B, and we can use B only with the "+" operator.

We denote by *RSC* (Repository Selected Components) the set of components selected form the repository. A component is added to solution if the component was not already used before and all the inputs of the component are provided for the tasks to be executed. A set of already used inputs is memorized (i.e. the number of already used inputs $noUsedInputs$, the set $setUsedInputs$ of already used inputs). A data dependency condition is also checked: the current added component must have all the data dependencies already satisfied. A solution is found when the last added component is a destination component. The solution of the backtracking algorithm contains components that are arranged such that the dependencies are satisfied.

An extra operation is applied to each solution to obtain the final solution with the "+" and "||" operators: the "+" operator is used only before components that depend on other components in the solution.

An example is provided in the thesis to discussed the proposed approach.

2.1.3 Temporal Components Composition Restraint algorithm

An algorithm that computes all possible component–based systems considering temporal dependencies between components is presented in what follows. Temporal composition restraints [Ves06, Ves07a] to assist the composition process (when selecting the next feasible component to integrate into the assembly) are introduced.

A top–down approach is used when reasoning about the way to solve a problem (from the system requirements develop the needed modules to accomplish the requirements of the under development system), and a bottom–up approach when assembling the pieces in order to build the final system.

The component composition is accomplished using a bottom-up approach: starting from a given set of components (stored in a repository) there are two main steps to obtain the final system: first, newly obtained components (if necessary) by assembling given components (simple components and/or compound components), and second, compose the final system from the new set of available components.

An issue concerning the composition process is the execution order of the involved components in the assembly. If there is a temporal restriction on the components execution order

then the final set of all possible configurations may be smaller. The necessity of user involvement to select the accurate resulted system from all generated configurations was discussed in [Ves06]. There are two different dependencies that must be taken into consideration when building all possible system configurations: data dependency and temporal dependency. The dependency conditions imply two composition restraints: data composition restraint and temporal composition restraint.

Definition 2.1.10 ([Ves07a]) **Data composition restraint** – *is a relation between two components in the composition that establishes that one component expects some results from the other component. Notation:* $CSender \xrightarrow{data} CReceiver$.

Definition 2.1.11 ([Ves07a]) **Temporal composition restraint** – *is a relation between two components that asserts that one component must be executed before the other component. Notation:* $CPrevious \xrightarrow{before} CAfter$.

The **TCCR** (**T**emporal **C**omponent **C**omposition **R**estraint) [Ves07a] algorithm uses a backtracking approach to generate all the component–based systems from the existing specified components taking into consideration the temporal composition restraints. The temporal restraints are explicitly specified in the algorithm input.

The used composition rules [Ves06] are: all ports (inports and outports) must be connected (no unused/lost data), one data provider is allowed for each inport of a component (one provider/inport), the data for all inports for every component must be provided, and the same data can be send ("broadcast") to more than one component to be used.

The first component that is used from repository is a source component. A component is added to the solution if it was not already used and all the inputs are provided [Ves07a]. Temporal composition restraints are also checked: between the current c component being checked and a previously added component cc no temporal dependency should exist $cc \xrightarrow{before} c$. A solution is found when the last added component is a destination component.

An example is given to understand and reason about our composition approach: a computation system for the Annual Student Average Mark. Each of the three disciplines have different mark computation rules. For each discipline a new compound component with different temporal restraints is developed. At the end, only three solutions (out of six possible) are valid.

2.2 Executing components configurations

The second step needed to be done when constructing systems out of components is to simulate systems execution and observe behavior. In what follows we present a minimalist execution model of component configurations.

2.2.1 Definitions and notations

The approach uses the specification from [FM04] and interface theories from [dAH01]. A new approach/improvement for specifying components is presented in this section.

Definition 2.2.1 ([Fan05]) *Input specification. The input* ***In*** *of a component function (or of the component) is specified using a 4-tuple of the form*

$$In = (name, type, semantic, value),$$

where: *name* represents the name of the inport, *type* represents the type of the inport, *semantic* describes the meaning of the inport, and *value* represents the value of the input at execution time.

Definition 2.2.2 ([Fan05]) *Output specification. The output* ***Out*** *of a component function (or of the component) is specified using a 4-tuple of the form*

$$Out = (name, type, semantic, value),$$

where: *name* represents the name of the outport, *type* represents the type of the outport, *semantic* describes the meaning of the outport, and *value* represents the value of the outport at execution time.

Definition 2.2.3 ([Fan05]) *Function specification. The function* ***FC*** *of a component is specified as follows:*

$$FC = (name, entryPoint),$$

where: *name* represents the name of the function and *entryPoint* represents the entry point of the function.

Definition 2.2.4 ([Fan05]) *Contract specification. The contract* ***cC*** *of a component (a function) is specified by describing the inputAssumption (specifies the value combinations at the input ports which the component must accept) and the outputGuarantee (specifies the value combinations the output ports of the component may produce).*

Definition 2.2.5 ([Fan05]) *Component specification. A simple component* ***C*** *is specified using a 5-tuple of the form*

$$C = (name, \{in\}, out, FC, cC),$$

where: *name* represents the name of the component, $\{in\}$ represents the inputs for the component function, *out* represents the output of the component function, FC represents the component function, and cC represents the component contract.

2.2.2 Construction and execution elements for a component–based model

The wiring of components in order to construct a component–based system is done using a connection between the output of a component and the input of another component.

Definition 2.2.6([Fan05]) *A connection K is made of an origin - output of a component, and a destination - input of another component:*

$$K = (origin, destination),$$

where: *origin* is an output of a component and *destination* is an input of a component.

The obtained assembly is also a component, i.e. a compound component using the [FM04] notation. The final system is called **BlackBox**. The following definition represents a more formal and detailed specification of the description from Subsection 2.1.2.

Definition 2.2.7 ([Fan05]) *A BlackBox is specified as follows:*

$$BlackBox = (\{in\}, out, \{component\}, \{connection\}),$$

where: $\{in\}$ represents the inputs for the blackbox, *out* represents the output of the blackbox, $\{component\}$ represents the components involved in the composition, and $\{connection\}$ represents all the connections between the involved components.

Definition 2.2.8 ([Fan05]) State of execution. At a given time of execution, the state is presented as follows:

$$State = (operation, componentForEval),$$

where:

- $operation = \{C \rightarrow, C =\}$;
  - $C \rightarrow$ represents propagation operation from the component C;
  - $C =$ represents evaluation operation of the component C.

• *componentForEval* is a component ready for evaluation.

The execution of the BlackBox component [Fan05] is a sequence of the form

$$(Op_0, C_0), (Op_1, C_1), (Op_2, C_2), ...$$

where for each $i \geq 0$, Op_i is a subset of possible operations and C_i is a subset of components ready for execution. Op_0 is a propagation operation and C_0 is a special "virtual" component called *start*.

The two possible operations are *propagation* and *evaluation*. The conditions for the propagation and evaluation rules are given and explained. If both types of operation may be performed, the propagation operation is chosen. Between many evaluation operations, one component is chosen randomly.

2.2.3 Algorithm for execution

The description of the algorithm for execution is described. We have used dynamic execution and reflection in Java. The subalgorithm (main method) for the execution of the BlackBox component is presented in the thesis. The execution of the model starts with the initialization, which consist of initialization for the input port of the *BlackBox* component. Propagation and/or evaluation are then applied.

2.2.4 Execution rules improvement

Using only the above rules we cannot model a repetitive or alternative execution. The repetitive and alternative structures are described in what follows. The rules are used during the assembly construction phase (only for the control flow design) and for the assembly execution.

Definition 2.2.9 ([Ves07b]) *Alternative structure - asks the execution of the CA component if the condition c is true, else asks the execution of the CB component.*

Definition 2.2.10 ([Ves07b]) *Repetitive structure - asks the execution of the CRepetitive component while the condition c is true. If the condition is false the execution of the CRepetitive component is finished and the execution continues with the CB component.*

Several examples are given in the thesis to exemplafy our approach.

2.3 Conclusions

Two algorithms for construction component configurations based on backtracking have been presented in this chapter: one that is based on input data dependencies and the other is based on temporal restraints. A component assembly execution model was presented. An extention of the previously defined execution model with new rules for execution is another contribution of this chapter. Some of the original results presented in this chapter have been reported in the papers [FM04, Ves06, Ves07a, Fan05, Ves07b].

Further work may be done in the following directions: generating the component configurations containing a selected tasks from an available set and checking if the constructed component configuration supports a given sequence of tasks.

3 Automata–based composition approach

Simple Component Selection Problem using automata representation is discussed in this chapter. Component system construction is done in two different ways: the first approach uses components and data dependencies between involved components and the second approach uses the task of the components. The first construction approach provides the data flow of the obtained system(s) and the second approach also provides the control flow.

3.1 Input data–based construction

In [PMP04] the component system is modeled as a finite automaton, where components are represented as states and information flows as transitions. The discussion from Section 1.3 about automata representation and the two approaches of component–based systems construction ([PMP04] and [MPP04]) revealed some drawbacks (disadvantages) of the second proposal. We have modified the algorithm from [PMP04] in order to generate all the nondeterministic finite automata. Also, the final constructed system have only live data. This property is checked after building the consistent system. The construction of the model is described in what follows.

3.1.1 MakeAllModels algorithm

The entities involved in the component system definition are presented in Section 2.1.

The **MakeAllModels** algorithm [FMD06] generates all the component–systems from a set of available and specified components. A backtracking algorithm is used. The obtained solutions are represented as automata.

The first component that is used in the component–based system is a source component (see Definition 2.1.1). A component is added to the solution if it was not already used and all the inputs of the component are provided for the tasks to be executed. A set of already used inputs is memorized (i.e. the number of already used inputs $noUsedInputs$, the set $setUsedInputs$ of already used inputs). A component configuration system is found when the last component added to the solution is a destination component (see Definition 2.1.2). The composition rules from Section 2.1.3 are used but using the automata representation.

The included examples in this thesis compute different configurations: the number of all solutions (no restrictions), the number of solutions without lost data, the number of solutions with only one provider/inport and the number of final solutions. The first example obtaines only one valid solution (with all the restrictions satisfied), and from the second set of available components no valid solution may be obtained.

3.2 Task–based construction

An algorithm that builds a finite automata–based model of a component–based system is proposed in [PMP04]. The algorithm checks the consistency of the model during its construction from a given set of components. We state before that the model from [PMP04] has some drawbacks regarding the solution given, as compared with the model from [MPP04]. In this section a new algorithm based on [MPP04] specification is presented, trying to overcome the disadvantages from the previous model [FMD06]. The basic steps are used but the integration of a component into the solution is different. The algorithm generates all possible configurations from a set of components.

A component is specified [MPP04] as follows:

Definition 3.2.1 ([MPP04]) *A component is specified by the following characteristics:* ***Component id*** *and* ***Interface****.*

A component may contain one or several class definitions, services, modules, even code, in a structural programming way (procedures, functions) a.s.o. The interface of the component should therefore, describe all the exported features of the component. The services provided by the component should be seen as functions, so the interface will specify a list of function signatures. We will conclude by saying that a component interface has the general form:

$$Interface = [Domain, List\ of\ signatures],$$

where the *Domain* describes the classes, modules and other data structures definitions provided by the component.

3.2.1 ControlFlow and DataFlow Syntactic Composition algorithm

This section contains the proposal of a new algorithm to construct all the component–based software systems as finite automata–based models. The obtained configurations are syntactically correct. By *syntactically correct configuration* we mean no semantic involvement, but just the way to connect the components together, the mechanical process of "wiring" components together (component integration).

Consider the component system $CS = \{C_1, C_2, ..., C_n\}$, in which every component C_i is specified as in the previous Definition 3.2.1. The first task that is used is a task of a source component (see Definition 2.1.1). The basic steps from the [FMD06] algorithm are applied but some changes are made. The final solution is composed not by the id of components but by the tasks. There are some extra properties that this algorithm has than the [FMD06]: the *tasks* from a component may be *executed at different execution times* and not sequential as in the previous model.

A task is added to solution if it was not already used and all the inputs of the task are provided. A component–based system is found when the last task added to solution is a task of a destination component (see Definition 2.1.2). The solution contains task from the set of available components. The solution array contains tasks and not components as in the previous proposal from Section 3.1.

Regarding computation of transitions there is an extra condition: for a transition $(mC, d) \rightarrow c$ it is not allowed in the solution to have a transition $(nC, d) \rightarrow c$, where mC and nC are from CS, with $mC \neq nC$. This condition assures that there is only one provider/port.

A major improvement from the previous two models [PMP04, MPP04] and the [FMD06] model is related to data flow and control flow: the algorithm provides also *the data flow* (by transitions) and *the control flow* (tasks to be executed, and in what order).

The data flow and control flow are both provided for the examples given in the thesis.

3.3 Conclusions

In this chapter we have proposed two approaches for component configurations construction using automata representation: one that is based on input data dependencies and the other is based on task dependencies. The second approach provides also the data flow and control flow. The chapter is based on the original work published in the papers [FMD06, VM06b, VM06a].

Further work can be done in the following directions: checking the behavior of components (to ensure that the assembled system does what is required) after syntactic composition and implementing a tool to build the syntactic model for a system of components and to analyze it based on its behavior.

4 Artificial intelligence–based composition approach

Genetic Algorithms and Genetic Programming–based approaches are used to investigate *Simple Component Selection Problem*. Each approach has different purpose: evolving components execution order and analyze the component composition. A component–based system is modeled using P–Systems. Rules for the Membrane Computing are applied.

4.1 Genetic Algorithms–based approach

4.1.1 Genetic Algorithms to generate Component Execution Order

A challenge in component–based software development is how to assemble components effectively and efficiently. A new approach to compute component execution orders using Genetic Algorithms (GA) is presented in this section.

Unlike previous work [FM04] which uses a backtracking algorithm to determine the execution order for system components, this proposal [FD05a] designs an execution order using evolutionary methods.

We want to find which component should be first executed and which is the order in which the components execute. We evolve arrays of integers which provide a meaning for executing the components within a system.

We denote this approach **E**volving **C**omponents **E**xecution **O**rder (ECEO). Each individual is a fixed-length string of genes. Each gene is an integer number, from 0 to $NumberOfComponents$. These values represent indexes order of the components. They will indicate the time moments for execution. Some components may be executed earlier and some of them are executed later. Therefore, a chromosome must be transformed so that it has to contain only the values from 0 to Max, where Max represents the number of different time moments (at one time moment it is possible that one or more components will be executed).

Several experiments are presented in the thesis. The number of evolved orders found with GA is compared to configurations number obtained with the backtracking algorithm from [FM04]. Numerical experiments show that the GA performs similarly and sometimes even better than standard backtracking approaches for several human-defined systems.

4.1.2 Genetic Algorithms to analyze Component Composition

In this approach we use the component model from [PMP04, FMD06]. The current [FD06b] proposal is obtained from an already developed automata-based model. We use the same features and properties from the given model but encoded into an evolutionary representation. The mapping steps from an automata–based model to a genetic–based model are given.

The proposed model uses a population of individuals, each individual being represented by a string of genes. The length of a string (or the chromosome length) is equal to the components number that is used by the system. Each gene corresponds to a certain component and it contains a number of ales equal to the number of outputs for corresponding component. Each ale represents the component index that uses the respective outport for the current component.

Numerical experiments are performed. For each experiment the best chromosome that encodes the optimal configuration is described. Also the corresponding automaton is presented.

4.2 Genetic Programming–based approach

4.2.1 Multi Expression Programming–based approach

Multi Expression Programming (MEP) is a Genetic Programming (GP) [Koz92, Koz94, GPW] variant that uses a linear representation of chromosomes. MEP individuals are strings of genes encoding complex computer programs. A unique MEP feature is its ability of encoding multiple solutions of a problem in a single chromosome.

Standard MEP [OD02, OGO04] algorithm uses steady–state evolutionary model [Sys89] as its underlying mechanism. The MEP genes are represented by substrings of variable length. The number of genes in a chromosome is constant and it represents the chromosome length. Each gene encodes a terminal (an element in the terminal set *T)* or a function symbol (an element in the function set *F*) [Fer01].

MEP is applied to determine optimal component combinations. Having the components involved in the composition for obtaining the complex system, we have applied MEP algorithm to components, where the operations are the two operations for component composition: parallel and serial composition. See Section 2.1.1 for details.

Our pattern is represented as a MEP component–based program whose elements are compound during the EA evolution. We have chosen Multi Expression Programming for representing the patterns because MEP provides an easy way to store a sequence of instructions (elements). In our approach, MEP stores only one solution (pattern) per chromosome.

Representation. The set of terminals are: $T = \{c_1, c_2, c_3, c_4, ..., c_n\}$, where c_i is the $i - th$ component and the set of primitive functions: $F = \{+, -\}$, where "+" represents a serial composition and "−" represents a parallel composition.

A didactical example is presented in the thesis.

4.2.2 CGP–based approach

This section contains an approach for behavior component composition using Cartesian Genetic Programming, enabling behavioral system composition from a set of specified components. The mapping elements from a system involving components to CGP, the representation and the algorithm are given in the thesis.

Cartesian Genetic Programming (CGP) [MT00] was introduced as an alternative methodology to standard Genetic Programming (GP), and secondly, to extend GP to non–Boolean problems and to show that it is a useful method for evolving programs with other data types [MT98, Mil99]. CGP is Cartesian in the sense that the method considers a grid of nodes that are addressed in a Cartesian coordinate system. The definition of a Cartesian Program (CP) is given in the thesis.

The definition of a CP and the definition of a component have given the idea that a system involving components might be generated using CGP, in which we have identified: the components are the nodes from the grid, and the components execution order, the execution rules and the component interconnections are incorporated in the genotype.

Given a set of components that are integrated into the final system we evolve the solution, taking into account the component composition behavior. Several types of experiments are performed: the first type uses a genotype-phenotype and requires that all nodes must be connected to each other and the final solution doesn't contain duplicate components, and the second type uses a genotype-phenotype and requires that all nodes are connected to each other, and we have multiplicity of components in the grid. The best chromosome that encodes the optimal configuration (for each experiment) is presented.

4.3 P–Systems–based approach

A new component–based model using P–Systems is presented in this section. The proposed approach is a technique that uses Membrane Computing to design a component–based model. The method uses symport/antiport rules to describe the computation of the involved components and an extra rule to transfer the result of a component to the inport of another. An example (with two model executions) reveals how to design a component–based system using P–Systems.

Membrane Computing (MC) [PRSZ02, Pau01b, Pau01a] is part of the powerful trend in computer science known under the name of Natural Computing. Its goal is to abstract computing models from the structure and the functioning of the living cells, as well as from the way the cells are organized. These models, called P–Systems, were investigated as mathematical objects, with the main goals of being a (theoretical) computer science type: computation power and usefulness in solving computationally hard problems. The paper [Pau00] was first circulated on web and comprehensive information may be found in the web page [PSy] or in [Pau02]. In MC area [FPPJ05] there are two main classes of systems: cell-like and tissue-like P–Systems. The former type is inspired from the cell organization, the latter one mimics the "collaboration" of cells from tissues of various kinds.

In [FD06a] we have used cell-like P–Systems with symport/antiport rules. The communication between compartments is done by means of uniport rules.

Subsection 4.3.1 contains the definition of a P–System and the description of the used rules: *symport rules* and *antiport rules*. We introduce an extra rule that performs for each P–Subsystem when no rules may be applied. This rule is placed in the output membrane of each P–Subsystem (component) and it fires when the computation in that component is finished. This rule is used for communicating the result of that component to the input of another component (P-Subsystem or the final result system).

In Subsection 4.3.2 a P–System encoding a composition process is described.

Subsection 4.3.3 contains an example of a component–based system designed as a P–System. The component–based system given as example decides if the number $6 * Nr1$ (with $Nr1$ given) is divided by the number $Nr2^2$ (with $Nr2$ given). The system has two inputs $Nr1$ and $Nr2$, and an output *yes* or *no*. The system is designed using P–Systems: the components are transformed into P–Subsystems and the communication of data between components is done using a special rule placed in the output membrane of each component. This rule is applied when the computation of the subsystem is finished. The initial configuration of the P–System and the rules are given. There are three components involved in the computation.

Subsection 4.3.4 describes the first model execution. We compute the result of the P–System with the input values $Nr1 = 5$ and $Nr2 = 4$. The result of the computation is *no* that means 30 is not divided by 16.

Subsection 4.3.5 described the second model execution. We compute the result of the P–System with the input values $Nr1 = 3$ and $Nr2 = 3$. The result of the computation is *yes* that means 18 is divided by 9.

4.4 Conclusions

Simple Component Selection Problem was investigated in this chapter. The proposed approaches are based on artificial intelligence techniques. A genetic algorithms–based approach is used to evolve component execution orders and to analyze component integration. A genetic programming–based approach is used to determine optimal component combinations using Multi Expression Programming and Cartesian Genetic Programming. A new approach that uses Membrane Computing (P–Systems.) to design a component–based system is proposed. The chapter is based on the papers [FD05b, FD05a, FD06b, FD06c, FD06a].

5 Multicriteria Component Selection Problem

The *Multicriteria Component Selection Problem* using various criteria is investigated in this chapter. A Greedy approach, a Branch and Bound and different Genetic Algorithms representations are proposed. The *Multicriteria Component Selection Problem* is modeled as a multiple objective optimization problem. Comparisons between the proposed approaches are discussed.

5.1 Greedy–based approach

In this section we address the problem of automatic component selection. In general, there may be different alternative components that can be selected, each coming at their own cost. We aim at a selection approach that guarantees the optimality of the generated component system, an approach that takes into consideration also the dependencies between components (view as restrictions on how the components interact).

We adapt a greedy approach by introducing a new selection function. The improved selection decision takes into account not only the cost of the components but also their interplay. The case study shows that considering the dependencies between the components the cost of the obtained solution may be higher due to the new selection improvement condition, but no dependencies between the selected components exist. In what follows we discuss the used selection functions.

Subsection 5.1.1 describes the Greedy selection functions: a ratio–based selection function and a component dependencies–based selection function.

The selection function is usually based on the objective function. We consider the proportion of number of requirements that the component c_i provides (from the set Remaind Set of Requirements – RSR) to the cost of the component as a measure to maximize our heuristic decision:

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

We denote this approach *GreedyR* from Greedy using the ratio selection function.

The interdependencies are an important factor when considering selection of a component from an available set. This first selection function does not consider the relations between requirements of the selected components.

The selection function [Ves08b] is augmented by using the dependencies between the selected components. To specify the component dependencies we introduce a dependency matrix for each component from the given set SC of components. We are only interested in the provided functionalities of the components that are in the set of requirements SR for the final system. We take into account only the dependencies between these requirements.

The new selection function takes into account the dependencies and selects the components such that $|SRc_i \bigcap RSR|/cost(c_i)$ *is maximal* and the number of dependencies of the component c_i considered *is minimal*. We denote this approach *GreedyRD* from Greedy using ratio and dependencies selection function.

5.1.2 Example

Starting for a set of six requirements and having a set of ten available components, 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.

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 1 the cost of each component from the set of components SC is presented.

Component	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 1: Cost values for each component in the SC

Table 2 contains for each component the provided services (in terms of requirements of the final system).

	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 2: Requirements elements of the components in SC

Table 3 contains the dependencies between each requirement from the set of requirements.

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		√				

Table 3: Specification Table of the Requirements Dependencies

The final solution (using the ratio–based selection) contains the components c_8, c_0, c_1 and c_7, components that satisfied all the requirements from the set of requirements SR. The cost of the final solution 36 is the sum of the cost of the selected components. The solution (using the component dependencies–based selection function) contains the following components: c_4, c_0, c_9 and c_1. The cost of the final solution 35, i. e. the sum of the cost of the selected components.

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 requirement, components and component costs and dependencies) the first approach may not be the correct solution do to the fact that the dependencies are not considered. The improvement of the selection function by using also the dependencies between the considered components helps us to compute the correct and accurate solution.

5.2 Branch and Bound–based approach

Subsection 5.2.1 contains the descriptions of the Branch and Bound functions. The main problem is what node for the list should be selected at a given moment in order to obtain the

shortest solution of the problem. Each node n from the list has an associated value (cost),

$$f(n) = g(n) + h(n),$$

where:

- $g(n)$ represents the cost of the components that were used until now (from the root node to node n) to construct the solution;
- $h(n)$ represents the number of remain requirements needed to be satisfied (to reach the final solution starting from the current node n). The function h is called heuristics function.

Subsection 5.2.2 contains the example from Subsection 5.1.2. Decimal scaling to normalize the cost of the components is used. Table 4 contains the normalization of the number of remain requirements to be satisfied.

No. of remaind requirements	Normalization	Value
0	0/6	0
1	1/6	0.16
2	2/6	0.33
3	3/6	0.50
4	4/6	0.66
5	5/6	0.83
6	6/6	1

Table 4: Normalization of the number of remaind requirements to be satisfied.

In what follows we discuss the application of the Greedy algorithm to our problem instance. The first solution description uses the selection function using the g and h functions descibed above. Our selection function considers the sum of number of remaind requirements to be satisfied and the cost of the already selected components plus the cost of the new selected component: $(g + h) is\ minimal$. The obtained solution contains the components: c_9, c_8 and c_0. The cost of the final solution 29 is the sum of the cost of the selected components. The second solution description uses the selection function using the g and h functions descibed above but also takes into consideration the dependencies. The obtained solution with all the requirements satisfied consists of the follosing components: $\{c_4, c_0, c_7, c_1\}$. The cost of the solution is 35.

Results obtained by Branch and Bound algorithm are described in what follows. The first solution description uses only the g and h function values to guide the selection. The obtained solution consist of: $\{c_8, c_6, c_2\}$ having the cost 0.28. In what follows the second method to select the best solution is described. It uses not only the function values of g and h but also the dependencies between components. The obtained solution taking into consideration also the component dependencies is: $\{c_4, c_2, c_6, c_1\}$. The cost of this solution is 34.

5.3 Genetic Algorithms–based approaches

5.3.1 Requirements–based chromosome representation

Multiobjective optimization problem using weighted sum method

In this proposal we approach the *Multicriteria Component Selection Problem* involving dependencies between components (requirements). We formulate the problem as multiobjective, involving 2 objectives and one constraint. The approach used is an evolutionary computation technique, a steady-state evolutionary model. The experiments and comparisons

with Greedy approach show the effectiveness of the proposed approach. There are several ways to deal with a multiobjective optimization problem. In this proposal the weighted sum method [Kd05] is used. We denote this approach $EArWS$ from the Evolutionary Algorithms requirements–based using weighted sum method. A more detailed discussion on the proposed approach may be found in [Ves08d].

A solution (chromosome) is represented as a string of size equal to the number of requirements from SR. The value of $i-th$ gene represent the component satisfying the $i-th$ requirement. The values of these genes are not different from each other (which means, same component can satisfy multiple requirements).

The used example is the one described in Section 5.1.2. Running the algorithm we could find more solutions (having the total cost 35) than the Greedy approach: [4 0 9 1 0 1], [4 9 9 1 0 9], [8 2 7 8 0 7], [4 0 5 5 0 1], [4 2 6 8 6 1], etc.

The result computed with $GreedyRD$ algorithm and with Evolutionary Algorithm (denoted $EArWS$ from Evolutionary Algorithm with requirements representation and that used weighted sum method) is stated in Table 5.

Algorithm		r_1	r_0	r_4	r_2	r_3	r_5	$cost$	no
$GreedyRD$		4	0	9	8	0	9	35	4
EArWS	best	8	2	6	8	6	6	28	3
	worst	4	2	7	8	0	9	55	6

Table 5: GreedyRD and EArWS Solutions

Multiobjective optimization problem using Pareto dominance principle

The *Multicriteria Component Selection Problem* is approached in this section. We formulate the problem as multiobjective, involving 2 objectives: the number of used components and the cost of the involved components. We use the Pareto dominance principle [AJG05] to deal with the multiobjective optimization problem.

In this proposal [Ves08e] we have not considered the dependencies because at end all the dependencies are satisfied due to the fact that all requirements must be satisfied and the dependencies are only between them. The approach used is an evolutionary computation technique, a steady-state evolutionary algorithm. The experiments and comparisons with the Greedy approach show the effectiveness of the proposed approach. We denote this approach as $EArP$ from Evolutionary Algorithm with requirements representation and that uses Pareto dominance principle.

A solution (chromosome) is represented as a string of size equal to the number of requirements from SR. The value of $i-th$ gene represent the component satisfying the $i-th$ requirement. The values of these genes are not different from each other (which means, same component can satisfy multiple requirements). The following two objective functions are considered: the total cost of the components used, $fCost$, and the number of components used, $fNoComp$.

A short and representative example is presented in what follows. Starting for a set of six requirements and having a set of twenty available components the goal is to find a subset of the given components such that all the requirements are satisfied. The set of requirements $SR = \{r_0,\ r_1,\ r_2,\ r_3,\ r_4,\ r_5\}$ and the set of components $SC = \{c_0, c_1,\ c_2,\ ...\ c_{19}\}$ are given. Table 6 contains for each component the provided services (in terms of requirements of the final system).

In the following we discuss the application of the Greedy algorithm using the first selection function from Section 5.1.2. The solution consists of five components and has the total cost

Component requirements	r_0	r_1	r_2	r_3	r_4	r_5	cost
c_0			√			√	25
c_1					√		7
c_2				√			3
c_3				√			3
c_4	√	√		√			17
c_5	√	√					12
c_6			√	√	√	√	37
c_7					√		5
c_8			√				27
c_9				√	√		8
c_{10}	√						15
c_{11}			√	√	√	√	34
c_{12}			√			√	25
c_{13}	√						13
c_{14}			√			√	26
c_{15}					√		6
c_{16}	√						13
c_{17}	√	√					12
c_{18}						√	28
c_{19}					√		5

Table 6: Requirements elements of the components in SC and the costs

71, with the representation: $[c_5,\ c_5,\ c_0,\ c_2,\ c_7,\ c_{14}]$.

Results obtained by the EArP approach are discussed in what follows. The parameters used by the evolutionary approach are as follows: mutation probability 0.7; crossover probability 0.7 and number of different runs 100. We have performed 3 different experiments considering different population sizes and different number of generations.

For the first experiment we have considered the following parameters: population size is 10 and number of iterations is 10. We can observe that in some situations we are obtaining 10 nondominated solutions which indicate that the whole final population is nondominated. Parameters used in the second experiment are as follows: population size 20 and number of iterations 20. For this experiment, we have obtained 20 nondominated solutions at the end of each run. Some of the solutions are similar, but it is important to note that all the solutions are finally becoming nondominated, which shows that a greater number of iterations and a bigger population size are conducting to better results. The third experiment performed has considered the following parameters: population size 50 and number of iterations 50. The number of nondominated solutions obtained in each of the 100 independent runs is 50.

While compared with the previous experiments we notice that we are getting a lower number of different solutions while cumulation the results obtained in all the 100 runs, but the quality of these solutions is improving much more while compared with first experiment and the second one. For instance, in the first experiment the greater value for the cost objective is 83 and in the second experiment is 71 while in the third experiment this is not more than 51. So, by increasing the number of iterations and the populations size we can observe that the diversity of the final solutions is decreasing but their quality is improving very much. But we should also mention that the best solution in terms of cost (which is 46) or number of components (which is 2) is obtained in all experiments.

The results computed with Greedy Algorithm and with Evolutionary Algorithm stated in Table 7. The last two columns contain the final cost and the number of used components in the presented solution.

Algorithm		r_1	r_0	r_4	r_2	r_3	r_5	*cost*	*no*
GreedyR		5	5	0	2	7	14	71	5
EArP	best	17	17	11	11	11	11	46	2
	worst	17	17	0	9	7	0	50	4

Table 7: GreedyR and *EArP* Solutions

5.3.2 Components–based chromosome representation

We formulate the problem as multiobjective, involving 3 objectives. The approach [VG08a] uses principles of evolutionary computation and multiobjective optimization [Gro05] combined with a greedy approach used to fine tune the solutions obtained by applying genetic operators (mutation and crossover). The experiments and comparisons with Greedy technique (*GreedyR* algorithm) show the effectiveness of the proposed approach. We denote this approach *EAcP* from Evolutionary Algorithm with chromosome representation and using Pareto dominance principle.

The following three objective functions are considered: the number of remain requirements to be satisfied, *fRemReq*; the total cost of the components used, *fCost*; the number of components used, *fNoComp*. All objectives are to be minimized. There are several ways to deal with a multiobjective optimization problem. In this proposal the Pareto dominance [AJG05] principle is used.

A solution (chromosome) [VG08a] is represented as a string of size equal to the number of components from SC. The value of i-th gene represents the set of requirements the component satisfies.

Solution fine tuning. An extra operation is used to fine tune the individuals after crossover and mutation. The fine tuning process consists of eliminating the requirements (in fact, the corresponding components) which are satisfied multiple times. We use a greedy–based heuristic to select the components that will have the requirements removed.

We have considered again the example from Section 5.3.1. The results obtained with the *GreedyR* algorithm is already presented. Results obtained by the Evolutionary Algorithm are presented in what follows. We have performed 3 different experiments considering different population sizes and different number of generations. For all experiments we used the same values for mutation and crossover probabilities which are: mutation probability 0.7; crossover probability: 0.7 and number of different runs 100.

For the first experiment we have considered the following parameters: population size: 10; number of iterations: 10. In some situation we are obtaining 10 nondominated solutions which indicate that the whole final population is nondominated. Parameters used in the second experiment are as follows: population size: 20; number of iterations: 20. The third experiment performed considers the following parameters: population size: 50; number of iterations: 50.

The approach [VG08a] described in this section combines the convergence efficiency of and evolutionary approach with a greedy approach whose role is to fine tune in an efficient way the solutions after the application of the genetic operators. For the hybrid evolutionary approach, one can deduce that we are obtaining better results with a smaller population and a smaller number of individuals which shows that these parameters are not influencing that much the final result.

5.3.3 Comparisons of the evolutionary Pareto dominance–based approaches

A short and representative example is presented in this section to help compare the requirements–based and components–based representations that use Pareto dominance principle. Starting for a set of six requirements and having a set of twenty available components the goal is to find a subset of the given components such that all the requirements are satisfied. The example is provided in details in the thesis.

We have performed different experiments [VG08b] considering different population sizes and different number of generations. For all experiments we used the same values for mutation and crossover probabilities which are: mutation probability 0.7; crossover probability 0.7 and number of different runs 100.

In order to compare 2 sets of nondominated solutions we introduce a measure. For 2 sets A and B of nondominated solutions, the dominance measure (denoted DM) gives the number of solutions from the set A which dominates solutions from the set B (denoted Dominates (A, B)) divided by the sum between number of different solutions (denoted $DifSol$) from A and the number of different solutions from B. Formally, the measure can be stated as follows.
Definition 5.3.2 ([VG08b]) Dominance measure (denoted DM)

$$DM(A,B) = \frac{\text{Dominates (A, B)}}{\text{DifSol(A)} + \text{DifSol(B)}}.$$

Similarly, we have:

$$DM(B,A) = \frac{\text{Dominates (B, A)}}{\text{DifSol(A)} + \text{DifSol(B)}}.$$

Remark 5.3.3 DM takes values between 0 and 1. The closer the value to 1 the better the results.

We have considered all the nondominated individuals obtained in all 100 runs.

For the first experiment we have considered the following parameters: number of generations: 10 and population size varying from 10 to 50 (we report the results obtained for 10, 20 and 50 individuals). A comparison between the two approaches in terms of DM measure is performed. The solutions obtained by the $EArP$ approach are better than the ones obtained by the $EAcP$ approach. It can also be deduced that the population size is not playing an important role in the evolution. Even with 10 individuals and 10 generations the results obtained are same like the ones obtained with 100 individuals in 10 generations.

For the second experiment we fix the population size at 10 and we let the number of generations to vary from 10 to 50. We report the results obtained for 10, 20 and 50 individuals. A comparison between the two approaches in terms of DM measure is performed.

From both experiments, $EArP$ is obtaining better results than EAcP [VG08b]. We should also mention that the solutions obtained by EArP for 10 iterations and 10 individuals are not improving neither by increasing the population size nor the number of generations. This means that the approach is performing very well even with these parameter values.

5.4 Conclusions

In this chapter we have proposed various approaches for construction component configurations using various criteria. A Greedy algorithm and two selection functions were proposed. Branch and Bound approach was used to solve the Multicriteria Component Selection Problem. Various genetic algorithms representations are discussed and compared. The chapter is based on the following papers that are published in international conferences [Ves08e, VG08a, Ves08c, VG08b, Ves08b, Ves08d].

6 Component Adaptation Architectures. A Formal Approach

In any component selection method, it is unrealistic to expect a perfect match between needed components and available components. The available components are adapted (adapting component signature properties, function adaptation, behavior adaptation) to meet the specific needs of the user. We discuss in this chapter function adaptation and four adaptation architectures are proposed. The behavior adaptation constraints for each architecture are given.

6.1 Correctness and component formal description

In a program P we distinguish [Fre05] three types of variables, grouped as three vectors X, Y and Z. The input vector $X = (x_1, x_2, ..., x_n)$ consists of the input variables. The input variables denote the known data of the given problem solved by the program P. We may suppose that they do not change during computation. The output vector $Z = (z_1, z_2, ..., z_m)$ consists of those variables which denote the results of the problem.

The specification of the program P is the pair formed from the input predicate $\varphi(X)$ and the output predicate $\psi(X, Z) : Specification : [\varphi(X), \psi(X, Z)]$.

A more formal component specification (than the ones described in the previous chapters) is given in what follows. A component specification is given by: the set of input variables - X; the set of output variables - Z; input predicate - $\varphi(X)$ and output predicate - $\psi(X, Z)$.
Definition 6.1.1 A Component specification is given by:
$ComponentName = [X = \{inputVar\}, Z = \{outputVar\}, \varphi(X), \psi(X, Z)]$.

6.2 Component function adaptation architectures

This section presents four adaptation architectures to be used when selecting components to obtain a larger system and there is a gap between the required and the provided functionalities in the selected components. There are cases when an adaptation of the involved components may provide the required functionality.

6.2.1 Adaptation architectures

The following adaptation architectures offer a new mechanism to compose different selected components in order to obtain a new component that fulfills the user requirements. Each new component is composed by using two selected components (except the repetitive adaptation architecture) and by taking into consideration the specification of the used components.

Serial (or sequential) adaptation architecture - SAA.

The serial adaptation architecture is based on the assumption that the output of the C_1 component is the input of the C_2 component. The behavior conditions state that for those values a of X for which the problem may be solved we have: the precondition of C_1 holds, the postcondition of C_1 makes the the precondition of C_2 to hold, and from the previous two conditions, the postcondition of C_2 will make the postcondition of the architecture to be true.

The new component composed using SAA from the two selected components is:

$$C_{CAA} = [\{i\}, \{o\}, \varphi_S(X), \psi_S(X, Z)],$$

where $\varphi_S : \varphi_1(i), i \in X_1$ *and* $\psi_S : \exists z \in Z_1 | \psi_1(i, z) \wedge \psi_2(z, o)$.

Parallel adaptation architecture - PAA.

The assumption of the parallel adaptation architecture is that the selected components involved in the composition satisfy only partially the user requirements. The union of the

outputs of the two selected components can satisfy the user requirements. The behavior conditions state that the user requirements (precondition) can be decomposed into two requirements (preconditions), and that the union of the two outputs of the selected components makes the postcondition of the adapted component to hold (for those values for which the problem may be solved).

The new component composed using PAA from the two selected components is:

$$C_{PAA} = [\{i_1, i_2\}, \{o_1, o_2\}, \varphi_P(X), \psi_P(X, Z)],$$

where $\varphi_P : \varphi_1(i_1) \wedge \varphi_2(i_2)$ *and* $\psi_P : \psi_1(i_1, o_1) \wedge \psi(i_2, o_2)$.

Alternative adaptation architecture - AAA.

The alternative adaptation architecture is based on the assumption that the precondition of the adapted components is only a part of the precondition of the user requirements. The requirements of the user are implemented using two separate cases. The behavior adaptation constraints state that: if the precondition of the adapted component is *true* then one precondition of the C_1 or C_2 components may be satisfied, if the precondition and postcondition of one of the components are satisfied then the postcondition of the adapted components will be *true*, and the last constraint, only one of the two preconditions of the two selected components may be *true* at the same time.

The new component composed using AAA from the two selected components is:

$$C_{AAA} = [\{i\}, \{o\}, \varphi_A(X), \psi_A(X, Z)],$$

where $\varphi_A : \varphi_1(i) \vee \varphi_2(i)$ *and* $\psi_A : \psi_1(i, o) \vee \psi_2(i, o)$, *where* $X = X_1 \cup X_2$ *and* $Z = Z_1 \cup Z_2$.

Repetitive adaptation architecture - RAA.

Let rc (*RepetitiveCondition*) be the repetitive condition and $bFalse$ and $bTrue$ values that will keep rc unmodified (in the current state). In the repetitive adaptation architecture an extra condition for the repetitive structure is provided. The behavior conditions state that if the precondition of the adapted component holds then the precondition of the selected component will also hold; if the precondition of the adapted component holds and the postcondition of the selected component is *true* and the extra condition rc is *true* then the precondition of the adapted component will remain *true*, and (the last behavior condition) if the precondition of the adapted component holds and the postcondition of the selected component is *true* and the extra condition rc does not hold then the postcondition of the adaptive component will be *true*.

The new component composed using RAA from the two selected components is:

$$C_{RAA} = [\{i\}, \{o\}, \varphi_R(X), \psi_R(X, Z)],$$

where $\varphi_R : \varphi_1(i), i \in X$ *and* $\psi_R : \exists o \in Z : \psi_1(i, o) \wedge \neg rc$.

6.3 Conclusions

Component function adaptation has been investigated in this chapter. When developing a system using pre–existing components the provided functionalities of the available set of components may not cover all the user requirements, the adaptation is used. *Serial (Sequential) Adaptation*, *Parallel Adaptation*, *Alternative Adaptation* and *Repetitive Adaptation* are the introduced adaptation architectures to provide a formal mathematical model for component function adaptation. Behavior adaptation constrains are discussed for each architecture. The chapter is based on the paper [Ves08a].

7 Metrics in Component–Based Software Engineering

Metrics have become essential in some disciplines of software engineering. Metrics are used for assessing the quality and complexity of software systems, as well as getting a basic understanding and providing clues about sensitive parts of software systems. In this chapter we adapt and propose metrics to quantify quality attributes of a component configuration. Metrics are also used to best select a component assembly from an available set.

7.1 New metrics to quantify quality attributes in Component–Based Software Engineering

The interaction among components in an assembly is essential to the overall quality of the system. When integrating components into a system assembly, it would be useful to predict how the quality attributes for the whole system will be. In order to predict and to asses quality attributes, the usage of software metrics is a necessity.

7.1.1 Formal Approach of Assembly

Definition 7.1.1 ([SV07b]) *An **assembly** is a binary relation denoted by $DR = (C, D), D \subseteq C \times C$, where C is a set of components and D is the relation that contains the dependencies between components. There is a component $c_0 \in C$ with a special role: to start the system execution.*

Definition 7.1.2 ([SV07b]) *A **dependency** is a pair $d = (c_1, c_2) \in D$ with the meaning that the execution of c_1 needs some services provided by c_2 (in other words, c_1 depends of c_2).*

We model a component–based system (an assembly of components) as a directed graph (DR) in which the vertices are the components (the set C) and the edges are the dependencies (the set D) between components. In this way we map each assembly to a directed graph.

Using directed graph view of the assembly is difficult to provide the depth and breath of the dependencies between involved components. A better view implies the transformation of the directed graph into a tree. The dependency tree construction [SV07b] is described in detail in the thesis ***D**ependency **T**ree **A**lgorithm* (**DTA**).

To obtained an optimal tree that contains all the paths from the directed graph representation, an additional algorithm that completes the tree is required. The proposed algorithm is called ***C**omplete **D**ependency **T**ree **A**lgorithm* (**CDTA**).

7.1.2 Adapted and defined metrics

In what follows the existing metrics from object–oriented design [LK94, CK93] that were adapted for component assemblies are presented.

In an object–oriented design, coupling is "the interconnectedness between its pieces" [CY91]. The declaration of an object of a remote class creates a potential collaboration between the two classes. This is measured by the metric CBO (Coupling Between Objects) [CK93].

We consider an assembly of components, $DR = (C, D)$, where C is a set of components and D is the relation that contains the dependences between components.

Definition 7.1.4 ([SV07b]) *A component c_1 **is coupled** with component c_2 if $(c_1, c_2) \in D$.*

Definition 7.1.5 ([SV07b]) **Coupling Between Components (CBC)** *of the component c metric is the number of components with which the component c is coupled.*

We are interested in coupling from the perspective of quality evaluation because an excessive coupling plays a negative role on many external quality attributes: *reusability, modularity, understandability* and *testability.*

Definition 7.1.6 ([SV07b]) **Depth Dependency Tree (DDT) of a component c_0 metric.** *Let us consider a component $c_n \in C$ and the corresponding elementary chain $c_0, c_1, ..., c_n$, where c_0 is the root node. The metric value of the c_0 component is $DDT(c_0) = n$. In other words, DDT measures the length chain dependencies from a given component to the root.*

Definition 7.1.7 ([SV07b]) **Breadth Dependency Tree (BDT)** *metric represents the number of chains dependencies from the root to all the leafs.*

We evaluate these metrics taking into account the impact on quality attributes. From this point of view, a high value of metric DDT makes the component hard to *reuse* in a different context. In addition, *understandability*, *maintainability* and *testability* are also affected. The understanding of an entity requires a recursive understanding of all the components that it depends. Moreover any change in a component c requires changes in all components that the c depends on. A high value of the metric affects *maintainability* and some of its criteria (*understandability*). The system tends to become increasingly complex.

Subsection 7.1.3 contains a system example, PDA - Personal Digital Assistant, to discussed the adapted and proposed metrics.

7.2 Metrics–based selection of a component assembly

The long–term success of Component–Based Development (CBD) depends on the ability to predict the quality of the obtained systems. For this reason researchers and practitioners are keen on developing techniques for efficient component selection and composition [CSSW04].

Software metrics that follow the assembly-centric evaluation approach are used to select (from all obtained assemblies) the solution that best represents the system requirements.

7.2.1 Proposed Metrics

We have proposed the following two metrics for measuring coupling between components.

Definition 7.2.1 ([SV07a]) **Component Coupling Grade.** *The Component Coupling Grade (CCG) of a component X which is dependent on a component Y, represents the number of services provided by Y that X uses. In what follows we will denote this value by CCG(X, Y).*

Definition 7.2.2 ([SV07b]) **Component Coupling Total Grade.** *The Component Coupling Total Grade (CCTG) of a component X which is dependent by a set of components C_1,C_2,...,C_n, represents the number of services provided by all these components that X uses.*

$$CCTG = CCG(X, C_1) + CCG(X, C_2) + ... + CCG(X, C_n). \quad (1)$$

7.2.2 The influence of metrics values on quality attributes

Our aim is to define metrics that are relevant in measuring the quality attributes which we are interested in. We need these informations for choosing the solution that best represents the system requirements.

The influence of metrics values on the quality attributes (whitch we consider important for the assembly evaluation) is presented in Table 8. We use the following notations: m for metric low value, M for hight value of the metric, "+" for positive influence and "−" for negative influence. For example a low value of IDC influences positively the reusability of the component.

A threshold is a limit (high or low) placed on a specific metric. All the above metric values scale between 0 and 1, except the CCTG and CCG. We set the value of the threshold at 0.5.

An example to discuss the proposed metrics is presented. In this example, nine components have been found as candidates. We add two more components to complete the final

	Reusability	Functionality	Understandability	Maintainability	Testability
PSU	m/+	m/-	m/+	m/+	m/+
RSU	m/+	-	m/+	m/+	m/+
CPSU	m/+	m/-	m/+	m/+	m/+
CRSU	m/+	-	m/+	m/+	m/+
IDC	m/+	m/-	m/+	m/+	m/+
IIDC	m/+	-	m/+	m/+	m/+
OIDC	m/+	m/-	m/+	m/+	m/+
AIDC	m/+	-	m/+	m/+	m/+
CCG	M/-	M/+	M/-	M/-	M/-
CCTG	M/-	M/+	M/-	M/-	M/-

Table 8: The influence of metrics values on quality attributes

system: a *Read* (R) and a $Write$ (W) component. The used algorithms [FMD06, VM06b] provided several solutions. We only present two solutions and discuss the different metrics values for each system–solution and their influences on the quality attributes.

The metrics values for the first solution are around the medium value, for all quality attributes. For example, the majority of the components have a very high functionality in the system and at the same time they can offer new functionalities for the future improvement by adding new provided services (influences the maintainability attribute). Regarding the coupling metrics we can remark that there is a maximum limit that is not very high and we may say that the maintainability and reusability are not strongly influenced. The assembly values metrics suggest that the solution is not considered to be the "best" for every quality attribute, but a medium "best" solution for the overall system. The value of the $AIDC$ metric is close to 1 but we must take also into consideration the $CCTG$ metric to decide which solution best represents our future needs (if we would like to improve and add new functionality or if we just want to have a good functionality for the system).

The second solution contains only three internal components from the set of candidate–components. The metrics values that influence the functionality attribute are close to 1 revealing a good functionality of each component inside the system, but the other metrics values influence negatively the other quality attributes. The 0.50 chosen threshold is exceeded for all the computed metrics. In Table 8 we can see that a high value influences negatively almost all the quality attributes discussed. The values of $CCTG$ metric are relatively high considering that there are few components in the solution. The $CCTG$ value for the ninth component is considered to be high yielding a very hard understandability, testability and maintainability.

7.3 Conclusions

In this chapter we have defined a set of metrics to quantify some quality attributes of component–based systems and a new set of metrics to select a component assembly (that best represents the system requirements) from a set of configurations. The chapter is based on the published papers [SV07b, SV07a].

8 Component Backtracking and Ant Colony–based approaches to solve TSP, Labyrinth Problem and AGAP

In this chapter we present three real problems that we have used to validate our previously described model: Traveling Salesman Problem, Labyrinth Problem and AGAP.

8.1 Backtracking and Ant Colony Component-based approaches to solve TSP

The *Traveling Salesman Problem* is designed using components in this section. The components may be reused to solve other problems, or to solve *Traveling Salesman Problem* using other techniques.

Subsection 8.1.2 presents the backtraking–based approach used to solve the TSP. The recursive and the sequential backtracking methods are used to show a comparative analysis of *Traveling Salesman Problem* models. The computational steps of the two models (recursive model and sequential model) are described using the execution model from Section 2.2.

Subsection 8.1.3 describes the component Ant (System) Colony–based approach to solve the TSP. The internal reasoning is presented: the iteration with all the ants and the usage of the global update rule, and the complete tour construction for an ant; the usage of the update local rule.

8.2 Component Ant Colony–based approach to solve Labyrinth Problem and AGAP

The Ant Colony approach is used to design the Labyrinth Problem and the Airport Gate Assignment Problem in this section.

Subsection 8.2.1 presents the architecture of the component–based approach for the Labyrinth Problem, the control flow and the data flow model. The solution using components is described: the first level of design contains initialization, computation and printing the obtained results. The next two levels are also presented: the second level contains the exploring ants , and the third level contains a more detail view of the PosPhAnts component (build a solution for an ant from the first colony). At the end of this subsection we have illustrated how the computation steps are successively executed.

Subsection 8.2.2 presents the architecture of the component–based approach for the Airport Gate Assignment Problem, the control flow and the data flow model. The solution using components is described: the first level of design contains initialization, computation and printing the obtained results, the second level contains the data generation for the greedy algorithm, the computation of the distance flow potentials and the ant system algorithm, and the third level contains a more detail view of the ant system algorithm computation (build solution, cost solution computation based on the constraints, best solution computation, and the update global rule computation). The steps of the computation of the ant colony component model for $AGAP$ are described.

8.3 Conclusions

Component–based models (a recursive and a sequential backtracking method) for solving the *Traveling Salesman Problem* are introduced. The developed models are used to solve Labyrinth Problem and AGAP using Ant Colony Systems. The chapter is based on the papers [FP05, VP07, PV07b, VP06a, PV07a, VP06b].

9 Conclusions and future work

The aim of the present PhD thesis was to be a supporting evidence to the idea that the component configurations construction is an important research and development area in Component–Based Software Engineering. The goal and objectives of this research are met as supported by this thesis. This thesis focuses on the activity of component integration and component composition. Construction component configurations by using different methods is one of the issues this thesis discusses.

A backtracking–based composition approach is one of the used approaches. Data and temporal component composition restraint configurations are proposed. A model for execution of a component assembly and new rules for execution are also developed.

Automata–based composition approach is another approach for component configurations construction. We have developed two different constructions: one is based on input data dependencies and the other is based on task dependencies.

Artificial intelligence–based approach is the third approach used for construction component configurations. Different intelligent–based approaches are used: Genetic Algorithms–based (to analyze component integration and to behavioral component composition) and Genetic Programming–based (to determine optimal component combinations and to design a component–based system using P Systems).

A Greedy approach, Branch and Bound approach and Genetic Algorithms approaches are used to develop solutions for the above stated problem but using various criteria. The used criteria are the total cost of used components and the number of used components.

A formal mathematical model for component function adaptation is proposed. Four component adaptation architectures for component functional adaptation are presented. The behavior adaptation constraints for each architecture (Serial or Sequential Adaptation Architecture, Parallel Adaptation Architecture, Alternative Adaptation Architecture and Repetitive Adaptation Architecture) are discussed.

A set of new metrics to quantify quality attributes of Component–Based Software Engineering and to best select a component assembly from a set of configurations are proposed in this thesis.

The developed models above are used to design a component–based system for the Traveling Salesman Problem, Labyrinth Problem and Airport Gate Assignment Problem using backtracking algorithms and Ant (Colony) Systems.

For each original approach we have suggested possible developments and new research directions in component configurations construction.

Bibliography

[AJG05] Ajith Abraham, Lakhmi Jain, and Robert Goldberg. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Springer Verlag, London, 2005.

[BA04] M. A. S. Boxall and S. Araban. Interface metrics for reusability analysis of components. In *Proceeding of Australian Software Engineering Conference ASWEC'2004*, pages 40–51, 2004.

[Bar05] A. Barton. A simplified ant colony system applied to the quadratic assignment problem. *Technical Report National Research Council of Canada no.47446*, pages 1–4, 2005.

[BBC05] Andrea Bracciali, Antonio Brogi, and Carlos Canal. A formal approach to component adaptation. *Journal of Systems and Software*, 74(1):45–54, 2005.

[BHSS06] Paul Baker, Mark Harman, Kathleen Steinhofel, and Alexandros Skaliotis. Search based approaches to component selection and prioritization for the next release problem. In *ICSM '06: Proceedings of the 22nd IEEE International Conference on Software Maintenance*, pages 176–185, Washington, DC, USA, 2006. IEEE Computer Society.

[Bre62] H. J. Bremermann. Optimization through evolution and recombination. In *Proceedings of the Conference on Self-Organizing Systems*, pages 93–106, 1962.

[BT85] E. Balas and P. Toth. *Branch and Bound Methods. In the Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. (E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, eds.) Wiley, 1985.

[CK93] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1993.

[CL02] I. Crnkovic and M. Larsson. *Building Reliable Component-Based Software Systems*. Artech House publisher, 2002.

[Crn03] I. Crnkovic. Component-based software engineering - new challenges in software development. In *Proceeding of the 25th International Conference on Information Technology Interfaces*, pages 9–18, 2003.

[CS01] P. T. Cox and B. Song. A formal model for component-based software. In *HCC '01: Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01)*, pages 304–311, 2001.

[CSSW04] I. Crnkovic, H. Schmidt, J. A. Stafford, and K. Wallnau. Message from the chairs. In *Proceedings of The* 6^{th} *ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction*, pages 1–7, 2004.

[CY91] P. Coad and E. Yourdon. *Object-Oriented Design*. Prentice Hall, 1991.

[dAH01] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In *Proceedings of the First International Workshop on Embedded Software*, pages 148–165, 2001.

[DDC99] M. Dorigo and G. Di Caro. *New ideas in optimization-The ant colony optimization meta-heuristic*. (D.Corne, M.Dorigo and F.Glover, eds.) London, McGraw-Hill, 1999.

[DDN00] S. Demeyer, S. Ducasse, and O. Nierstrasz. Finding refactorings via change metrics. In *In Proceedings of OOPSLA 2000, ACM SIGPLAN Notices*, pages 166–178, 2000.

[DG97] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evol. Comp.*, pages 53–66, 1997.

[DLRZ04] H. Ding, A. Lim, B. Rodrigues, and Y Zhu. The airport gate assignment problem. In *Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS'04)*, pages 74–81, 2004.

[Dor92] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartamento di Elettronica, Politecnico di Milano, 1992.

[DW98] D. F. D'Souza and A. C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, Reading, MA, 1998.

[Fan05] **A. Fanea**. Specification, construction and execution of a component-based model. In *Proceedings of the Cluj Computer Science Academic Colloquium*, pages 87–92, 2005.

[FBPFR04] Michael Roy Fox, David C. Brogan, and Jr. Paul F. Reynolds. Approximating component selection. In *WSC '04: Proceedings of the 36th conference on Winter simulation*, pages 429–434. Winter Simulation Conference, 2004.

[FD05a] **A. Fanea** and L. Diosan. Components execution order using genetic algorithms. *Studia Universitas Babes-Bolyai, Seria Informatica*, L(2):23–32, 2005.

[FD05b] **A. Fanea** and L. Diosan. Designing a component-based machine using multi expression programming. In *Proceedings of the Cluj Computer Science Academic Colloquium*, pages 93–98, 2005.

[FD06a] **A. Fanea** and L. Diosan. Component based model using p systems. *The International Journal of Information Technology and Intelligent Computing, ISSN: 1895-8648*, 1(3):499–508, 2006.

[FD06b] **A. Fanea** and L. Diosan. Computational intelligence-based model for component composition analysis. In *Proceeding of the International Conference on Computers, Communications and Control, ISSN: 1841-9836*, pages 474–479, 2006. (indexed ISI-SCI-E)

[FD06c] **A. Fanea** and L. Diosan. Evolutionary approach for behaviour component composition. In *Proceeding of the International Conference on Computers, Communications and Control, ISSN: 1841-9836*, pages 480–485, 2006. (indexed ISI-SCI-E)

[Fer01] C. Ferreira. Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems*, 13:87–129, 2001.

[Flo67] R.W. Floyd. Assigning meaning to programs. In *Proceedings of Symposium on Applied Mathematics*, pages 19–32. J.T. Schwartz (Ed.), A.M.S., 1967.

[FM04] **A. Fanea** and S. Motogna. A formal model for component composition. In *Proceedings of the Cluj Academic Days National Symposium*, pages 160–167, 2004.

[FMD06] **A. Fanea**, S. Motogna, and L. Diosan. Automata-based component composition analysis. *Studia Universitas Babes-Bolyai, Seria Informatica*, LI(1):13–20, 2006.

[FP05] **A. Fanea** and C. M. Pintea. A component based-model for a np-hard problem. *Annals of Oradea University, Fascicola Matematica*, XII(0):91–100, 2005.

[FPPJ05] R. Freund, G. Paun, and M. J. Prez-Jimnez. Tissue p systems with channel states. *Theoretical Computer Science*, 330:101–116, 2005.

[FPS06] F. Frentiu, H. F. Pop, and G. Serban. *Fundamentals of Programming*. Cluj University Press, 2006.

[Fre05] Militon Frentiu. Correctness: a very important quality factor in programming. *Studia Universitas Babes-Bolyai, Seria Informatica*, L(1):11–20, 2005.

[GA05] M. A. Goulo and F. B. Abreu. Composition assessment metrics for cbse. In *Proceedings of The 31st Euromicro Conference, Component-Based Software Engineering Track*, pages 96–103, 2005.

[GBV06] Guillaume Grondin, Noury Bouraqadi, and Laurent Vercouter. Madcar: An abstract model for dynamic and automatic (re-)assembling of component-based applications. In *CBSE*, volume 4063 of *Lecture Notes in Computer Science*, pages 360–367. Springer, 2006.

[GG07] Lars Gesellensetter and Sabine Glesner. Only the best can make it: Optimal component selection. *Electron. Notes Theor. Comput. Sci.*, 176(2):105–124, 2007.

[Gol89] D. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[GPW] www.genetic-programming.com.

[Gro05] C. Grosan. *A comparison of several evolutionary models and representations for multiobjective optimization*. ISE Book Series on Real Word Multi-Objective System Engineerin, chapter 3, Nova Science,, 2005.

[HDM03] A. v. d. Hoek, E. Dincel, and N. Medvidovic. Using service utilization metrics to assess and improve product line architectures. In *In Proceedings of the 9th IEEE International Software Metrics Symposium Metrics*, page 298, 2003.

[Hem05] David Hemer. A formal approach to component adaptation and composition. In *ACSC '05: Proceedings of the Twenty-eighth Australasian conference on Computer Science*, pages 259–266, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[Hen00] T. Henzinger. Masaccio: A formal model for embedded components. In *Proceedings of the First IFIP*, pages 549–563, 2000.

[HMH+07] Nima Haghpanah, Shahrouz Moaven, Jafar Habibi, Mehdi Kargar, and Soheil Hassas Yeganeh. Approximation algorithms for software component selection problem. In *APSEC*, pages 159–166. IEEE Computer Society, 2007.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[ITV02] L. Iribarne, J. Troya, and A. Vallecillo. Selecting software components with multiple interfaces. In *In Proceeding of the 28th EUROMICRO Conference Component-Based Software Engineering*, pages 26–32. IEEE Computer Society, 2002.

[Kd05] Y. Kim and O.L. deWeck. Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. *Structural and Multidisciplinary Optimization*, 29(2):149–158, 2005.

[Kni99] Günter Kniesel. Type-safe delegation for run-time component adaptation. In *ECOOP '99: Proceedings of the 13th European Conference on Object-Oriented Programming*, pages 351–366, London, UK, 1999. Springer-Verlag.

[Koz92] R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. AMIT Press, 1992.

[Koz94] R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. AMIT Press, 1994.

[LK94] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics*. Prentice-Hall Object-Oriented Series, Englewood Cliffs, 1994.

[MA03] Brandon Morel and Perry Alexander. Automating component adaptation for reuse. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, page 142, 2003.

[Men32] K. Menger. Das botenproblem. *Ergebnisse Eines Mathematischen Kolloquiums 2*, pages 11–12, 1932.

[Mil99] J. F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *In Proceedings of the 1st Genetic and Evolutionary Computation Conference*, pages 1135–1142, 1999.

[Mor04] Brandon Morel. Spartacas automating component reuse and adaptation. *IEEE Trans. Softw. Eng.*, 30(9):587–600, 2004. Senior Member-Perry Alexander.

[MP02] S. Motogna and B. Parv. A formal model for components. *Bul. Stiint. Univ. Baia Mare, Seria B, Matematica-Informatica*, XVIII:269–274, 2002.

[MPP04] M. Motogna, D. Petrascu, and B. Parv. Automata-based compositional analysis of component systems. design and implementation issues. In *Proceedings of the Fourth International Conference on Applied Mathematics (ICAM4)*, pages 197–203, 2004.

[MT98] J. F. Miller and P. Thomson. Aspects of digital evolution: Evolvability and architecture. In *Proceedings of the Parallel Problem Solving from Nature*, pages 927–936, 1998.

[MT00] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Proceedings of the European Conference on Genetic Programming*, pages 121–132, 2000.

[MTF97] J. F. Miller, P. Thomson, and T. Fogarty. *Electronic Circuits using Evolutionary Algorithms. Arithmetic Circuits: A Case Study.* Wiley, 1997.

[Nad02] D. Naddef. *Polyhedral Theory and Branch-and-Cut Algorithms for the Symmetric TSP. In The Traveling Salesman Problem and its Variations.* (G.Gutin and A.P.Punnen, eds.) Kluwer, 2002.

[NH04] V. L. Narasimhan and B. Hendradjaya. A new suite of metrics for the integration of software components. In *The First International Workshop on Object Systems and Software Architectures WOSSA 2004*, 2004.

[OD02] M. Oltean and D. Dumitrescu. Multi expression programming. Technical report, Babes-Bolyai University, 2002.

[OGO04] M. Oltean, C. Grosan, and M. Oltean. Evolving digital circuits for the knapsack problem. In *Proceedings of the International Conference on Computational Sciences, E-HARD Workshop*, pages 1257–1264, 2004.

[Pau00] G. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 2000.

[Pau01a] G. Paun. From cells to computers: Computing with membranes. *BioSystems*, 59:139–158, 2001.

[Pau01b] G. Paun. P systems with active membranes: Attacking np-complete problems. *Journal of Automata, Languages and Combinatorics*, 6:75–90, 2001.

[Pau02] G. Paun. *Computing with Membranes: An Introduction.* Springer-Verlag New York, 2002.

[PD05] C.-M. Pintea and D. Dumitrescu. Improving ant systems using a local updating rule. In *Proc. 7-th Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 295–299, 2005.

[PD07] C.-M. Pintea and D. Dumitrescu. Introducing pharaoh ant system. In *Proceedings of MENDEL 2007, the 13th International Conference on Soft Computing*, pages 54–59, 2007.

[PMP04] B. Parv, S. Motogna, and D. Petrascu. Component system checking using compositional analysis. In *Proceedings of the International Conference on Computers and Communications*, pages 325–329, 2004.

[PPPJRP05] G. Paun, J. Pazos, M. J. Perez-Jimenez, and A. Rodriguez-Paton. Symport/antiport p systems with three objects are universal. *Fundamenta Informaticae*, 64:353–367, 2005.

[PRSZ02] G. Paun, G. Rozenberg, A. Salomaa, and c. Zandron. Membrane computing. In *International Workshop, WMC-CdeA*, 2002.

[PSy] The p systems web page. http://psystems.disco.unimib.it.

[PV07a] C.-M. Pintea and **A. Vescan**. Component-based ant system for a biobjective assignment problem. *Studia Universitas Babes-Bolyai, Seria Informatica e*, LII(1):21–32, 2007.

[PV07b] C.-M. Pintea and **A. Vescan**. The labyrinth problem: Component model with pharaoh system. In *Proceedings of International Conference on Fundamental Science, ICFS 2007, Applied mathematics and computer science section, ISBN: 978-973-759-367-2*, pages 82–86, 2007.

[SB03] D.J.T. Sumpter and M. Beekman. From nonlinearity to optimality: pheromone trail foraging by ants. *Anim. Behav.*, 66:273–280, 2003.

[SBF99] T. R. Stickland, N. F. Britton, and N. R. Franks. Information processing in social insects. *C.Detrain, J.L.Deneubourg, J.M.Pasteels, eds.*, pages 83–100, 1999.

[SDS04] A. Syropoulos, S. Doumanis, and K. T. Sotiriades. Computing recursive functions with p systems. In *Pre-proceedings of the Fifth Workshop on Membrane Computing (WMC5*, pages 414–421, 2004.

[SP04] G. Serban and C. M. Pintea. Heuristical and learning approaches for the traveling salesman problem. *Studia Universitatis Babes-Bolyai, Informatica*, XLIX:27–36, 2004.

[SV07a] C. Serban and **A. Vescan**. Metrics-based selection of a component assembly. *Special Issue of Studia Universitatis Babes-Bolyai Informatica: Proceeding of The International Conference on Knowledge Engineering: Principles and Techniques, ISSN: 1224-869X*, pages 324–331, 2007.

[SV07b] C. Serban and **A. Vescan**. Metrics for component-based system development. *Creative Mathematics and Informatics, ISSN: 1843-441X*, 16:143–150, 2007.

[Sys89] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, 1989.

[Sys91] G. Syswerda. A study of reproduction in generational and steady state genetic algorithms. In G. J. E. Rawlins, editor, *Proceedings of FOGA Conference*, pages 94–101. Morgan Kaufmann, 1991.

[Szy98] C. Szyperski. *Component Software, Beyond Object-Oriented Programming.* ACM Press, Addison-Wesley, 1998.

[Ves06] **A. Vescan**. Restraint order component model execution. In *The 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, ISBN:978-0-7695-2740-6*, pages 195–200, 2006. (ISI Proceeding)

[Ves07a] **A. Vescan**. Components ordered assembly construction based on temporal restraint. In *Proceeding of the 3rd Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, ISBN: 978-80-7355-077-6*, pages 249–256, 2007.

[Ves07b] **A. Vescan**. Modeling component compositions and assembly execution. In *Proceeding of the Second Annual International Conference of Students, Postgraduates and Young Scientists, Computer Science and Engineering - 2007, ISBN: 978-966-553-649-9*, pages 20–24, 2007.

[Ves08a] **A. Vescan**. Component adaptation architectures. a formal approach. In *The 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, ISBN: 978-3-540-85566-8*, pages 319–326, 2008. (ISI Proceeding)

[Ves08b] **A. Vescan**. Dependencies in the component selection problem. In *Proceedings of the 6th ICAM - International Conference on Applied Mathematics*, page (accepted), 2008.

[Ves08c] **A. Vescan**. An evolutionary multiobjective approach for the component selection problem. In *Proceedings of the First IEEE International Conference on the Applications of Digital Information and Web Technologies, ISBN: 978-1-444-264-9* , pages 252–257, 2008. (indexed IEEE)

[Ves08d] **A. Vescan**. Optimal component selection using a multiobjective evolutionary algorithm. *Neural Network world, ISSN: 1210-0552* , page (accepted), 2008. (indexed ISI-SCI-E)

[Ves08e] **A. Vescan**. Pareto dominance - based approach for the component selection problem. In *Proceedings of the 2nd UKSim European Symposium on Computer Modeling and Simulation, ISBN: 978-0-7695-3325-4* , pages 58–63, 2008. (indexed IEEE)

[VG08a] **A. Vescan** and C. Grosan. A hybrid evolutionary multiobjective approach for component selection problem. In *Proceedings of the 3rd International Workshop on Hybrid Artificial Intelligence Systems, ISBN: 978-3-540-87655-7* , pages 164–171, 2008. (ISI Proceeding)

[VG08b] **A. Vescan** and C. Grosan. Two evolutionary multiobjective approach for component selection problem. In *Proceedings of the Fourth International Workshop on Evolutionary Multiobjective Optimization - Design and Applications* , page (accepted), 2008. (indexed IEEE)

[VGP08] **A. Vescan**, C. Grosan, and H. F. Pop. Evolutionary algorithms for the component selection problem. In *Proceedings of the 2nd International Workshop on Evolutionary Techniques in Data Processing, ISBN: 1529-4188* , pages 509–513, 2008. (ISI Proceeding)

[VM06a] **A. Vescan** and S Motogna. Syntactic analysis of component composition. *Pure Mathematics and Applications (PUMA)*, 17(3-4):527–537, 2006.

[VM06b] **A. Vescan** and S. Motogna. Syntactic automata-based component composition. In *The 32nd EUROMICRO Software Engineering and Advanced Applications (SEAA), Proceeding of the Work in Progress session, ISBN: 3-902457-11-2*, pages 13–14, 2006.

[VP06a] **A. Vescan** and C.-M. Pintea. Ant system. a component based-model. In *Proceeding of the 2nd International Conference on Intelligence Computer Communication and Processing, ISBN: 973-662-235-5* , pages 23–28, 2006. (indexed IEEE)

[VP06b] **A. Vescan** and C.-M. Pintea. Comparative models for a combinatorial problem. In *Proceeding of The 6th International Conference Communication, ISBN: 978-973-718-479-5* , pages 253–256, 2006. (indexed IEEE)

[VP07] **A. Vescan** and C.-M. Pintea. Ant colony component-based system for traveling salesman problem. *Journal of Applied Mathematical Science, ISSN: 1312–885X*, 1(25–28):1347–1357, 2007.

[VP08] **A. Vescan** and H. F. Pop. The component selection problem as a constraint optimization problem. In *Proceedings of the Work In Progress Session of the 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques*, page (accepted), 2008.

[WYF03] H. Washizaki, H. Yamamoto, and Y. Fukazawa. A metrics suite for measuring reusability of software components. In *In Proceedings of 9th IEEE International Software Metrics Symposium METRICS 2003*, pages 211–223, 2003.

[XB01] J. Xu and G. Baile. The airport gate assignment problem, mathematical model and a tabu search algorithm. In *Proc. of the 34th Annual Hawaii International Conference on System Sciences*, pages 1–10, 2001.

[XW05] Xie Xiong and Zhang Weishi. The current state of software component adaptation. In *SKG '05: Proceedings of the First International Conference on Semantics, Knowledge and Grid*, page 103, Washington, DC, USA, 2005. IEEE Computer Society.