Specification-based Retrieval of Software Components through Fuzzy Inference

Ioana Şora, Doru Todinca

Department of Computer Science, "Politehnica" University of Timişoara
Bd. V. Parvan 2, 300223 Timisoara, Romania
E-mail: {ioana.sora, doru.todinca}@ac.upt.ro

Abstract: In the component based software engineering approach, a software system is viewed as an assembly of reusable independently developed components. In order to produce automated tools to support the selection and assembly of components, rigorous specifications of components and performant retrieval and selection strategies based on these specifications are needed. Classical approaches for automatically retrieving components that match a set of required properties result mostly in very complex solutions. In this article we propose an efficient fuzzy logic based solution for the specification and retrieval of software components. We describe the basic principles of the proposed solutions and illustrate them on example scenarious.

Keywords: fuzzy logic, fuzzy inferece, software components

1 Introduction

In the component based software engineering approach [1], [11], a software system is viewed as an assembly of reusable independently developed components. The set of components and the manner in which these are connected with each other determines the properties (functionality and behaviour) of the assembled system.

Constructing a system with certain required properties starts with the compositional decision: which components to select from a large repository of components and which topology to give to their assembly? During this decisional process, the selection is made based on the matching between the required properties and the known properties of the components. In order to use automatic tool support, a systematic compositional model is needed. Such a compositional model must comprise a component specification scheme and formalism, and a coordinated, well defined requirements driven composition strategy. It must have the expresiveness to describe many types of requirements and properties (functional, non-functional, structural, behavioural), it must be sufficiently formal

to be used in automatic tools and still its complexity should permit reasonable implementation.

Many different approaches have been proposed for the specification of components: interface description, formal specification methods, behavioural specification, architectural description. In previous work, we also developed a compositional model at the architectural level, the composable components model and its specification formalism, CCDL [8], [10]. A critical issue are the non-functional properties of components, as speed, memory consumption, usability, etc. These are difficult to specify and match [3], most often the matching of a set of such requirements is based on a series of trade-offs. This work proposes a solution by applying fuzzy logic in this field of software engineering.

The remainder of this article is organized as follows: Section 2 resumes the basic concepts of our component model, CCDL. Section 3 presents the extension of CCDL with fuzzy attributes and fuzzy logic based selection of components. All concepts are illustrated on a running example, built incrementally over all sections. Section 4 discusses our work in the context of other works. The last section summarizes the concluding remarks.

2 Core CCDL Component Model

This section resumes the basic concepts of our component model. Only these aspects of the complex composable components model (CCDL) that are dirrectly affected by the extension proposed in this work are presented here, many details beeing omitted. The CCDL model has been extensively presented in [8] and [10].

2.1 Principles Used in the Specification of Components

A software system is viewed as a set of components that are connected to each other through connectors. As defined in mainstream component bibliography [11], [1] a software component is an implementation of some functionality, available under the condition of a certain contract, independently deployable and subject to composition.

In our approach, each component has a set of ports as logical points of interaction with its environment. We distinguish between input ports and output ports and consider that syntactically every input port is plug-compatible with every output port. The logic of a component composition (the semantic part) is enforced through the checking of component contracts.

Components may be simple or composed. A simple component is the basic unit of composition that is responsible for certain behaviour and has one input port and

one output port. Composed components introduce a grouping mechanism to create higher abstractions and may have several input and output ports.

Components are specified by means of their provided and required properties. Properties in our approach are facts known about the component, in a way similar to Shaw's credentials [7]. A property is a name from a domain vocabulary set and may have refining subproperties (which are also properties) or refining attributes that are typed values. For example, a property that does data compression will be described through a property named compression. An attribute of this property can be defined as the average compression rate, expressed as the real type attribute compression-factor.

The component contracts specify the services provided by the component and their characteristics on one side and the obligations of client and environment components on the other side. Most often the provided services and their quality depends on the services offered by other parties, being subject to a contract. In the CCDL model contracts are expressed as provides and requires clauses containing sets of provided and required properties. The component as a whole provides certain services, defined by global provides clauses in the component specification. In order to provide these services it requires that other services are provided by the environment. These services must be provided in certain defined interaction points, thus the requires clauses are attached to the ports.

2.2 Automatic Requirements-driven Selection and Composition

A component assembly is valid if it provides all user required services and if the contracts of all individual components are respected. A contract for a component is respected if all its required properties have found a match. The criterion for a semantically correct component assembly is matching all required properties with provided properties on every flow in the system.

A match between a required and a provided property is established first by matching the properties names, then recursively matching subproperties and matching of the attributes values. A property present in the requirement must not specify all attributes of the property present in the provides clause for a match.

In our approach, it is not necessary that a requirement of a component is matched by a component dirrectly connected to it. It is sufficient that requirements are matched by some components that are present on the flow connected to that port, these requirements are able to propagate.

The internal structure of a composable target can be established at runtime through automatic requirements-driven composition. The requirements for the composable target result from its invariant structural constraints and from the current requirements imposed by the external environment. The overall process of

generating the structure of the target is driven by the requirements. The required properties for the target are put on the main flow of the target and propagated from that point on, while adding components. The addition of new components on the flow occurs according to the current requirements, which are those propagated from the initial requirements together with those of the new introduced components. A component is added to the solution if it matches at least a subset of the current requirements. A solution is considered complete when the current requirements set becomes empty. It is possible that for a certain set of requirements no solution can be found or that several component assemblies are found.

The mechanism of propagation of requirements briefly sumarised here was formally described in [9], paper that also gives a complete description of the automatic composition strategy.

2.3 Issues

The compositional model presented above can be improved in regard with the specification and matching of properties and attributes.

In the core CCDL model, a property most often consists of a name describing functionality and attributes that are typed values describing the non-functional characteristics of the functional property.

For example, a component providing data compression can be characterized by a property named compression. The non-functional characteristics are described by attributes like compression_rate, speed, memory_consumption. For the specification of the component, these attributes must be first evaluated. The compression_rate can be expressed as a real type value, representing the average measured compression rate for different data inputs. Not all attributes can be as easily evaluated and described as crisp values. The exact value of the speed attribute depends essentially on the hardware configuration of the system. Such an attribute should be more adequately be described using terms as ‘fast’, ‘very fast’, ‘slow’, description established as a result of comparing the performance of the current component relative to the performance of other components that provide the same functional property.

In the core CCDL model, a match between a required and a provided property is established first by matching the properties names, then recursively matching subproperties and matching of the attributes values. Let consider components C1 and C2. C1 provides property P with the attributes A1=Value1, A2=Value2. C2 provides the same property P with the attributes A1=Value3, A2=Value4. A client requirement can be for property P with attributes A1=ValueX, A2=ValueY. Then either C1 or C2 will be selected, whether ValueX=Value1 and ValueY=Value2 is true or ValueX=Value3 and ValueY=Value4. If the values do not match exactly, no component will be selected.

In many cases, a client requirement has no precise requirements regarding the values of attributes. For example, a requirement regarding the selection of a compression component will rarely need to specify an exact value for the compression_rate as to require, i.e., compression_rate=0.4. It will rather specify that compression_rate=medium is acceptable.

In order to relax the specification of requirements, in the strict matching mechanism of the core CCDL it is still possible that a property present in the requirement does not have to specify all attributes of the property present in the provides clause for a match. In the previous example of components C1 and C2, a client requirement can omit the specification of required values for all or some of the attributes of property P. For example, a requirement as property P with attribute A2=ValueX will select all components that provide property P with attribute A2 matching the ValueX, regardless of the values of the attribute A1. This possibility of uncomplete requirement specification is not enough to give the desired flexibility. It misses the possibility to specify required ranges, as well as the possibility of doing tradeoffs between the matching degree of several attributes of the same property.

From the examples presented above, we conclude that there are two main issues with the specification and matching of properties:

- certain attributes cannot be exactly evaluated and specified
- the matching of required/provided attributes must not always be precise

We believe that fuzzy logic [13] can help to overcome these problems because it aims at exploiting the tolerance for imprecision and uncertainity. The next chapter proposes our fuzzy based solution.

3 Fuzzy Attributes and Selection

In this article, we propose an extension of the core CCDL model, where attributes can be expressed in fuzzy logic and the properties matching is done by fuzzy inference.

3.1 Fuzzy Attributes

3.1.1 Concepts

A property consists of a name describing functionality and attributes that are either typed values (crisp) or fuzzy terms.

For example, the property compression can be defined with attributes that are both crisp and fuzzy. The atributes compression_rate and memory_consumption are crisp values, the attribute speed is a fuzzy value as it depends on the hardware configuration and performance of the system.

The names used for the properties and for the attributes are established through a domain-specific vocabulary. Such a restriction is necessary because a totally free-text specification makes the retrieval difficult, producing false-positive or false-negative matchings due to the use of a non-standard terminology. Establishing domain specific vocabularies is a common solution [6], [5] and has been used also in the core CCDL model [8].

In our work, the domain specific vocabulary must also describe the domains of the fuzzy attributes (linguistic variables) for each property as well as the membership functions for the fuzzy terms.

The membership functions for all linguistic variable are considered of trapezoidal/triangular shape as in Figure 1:

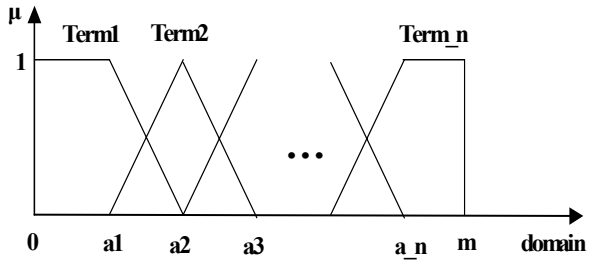

Figure 1
The shape of the membership functions

For each linguistic variable, first the number and the names of the terms of its domain must be declared, and after that the values of the parameters a1, a2, …, a_n must be specified.

3.1.2 Example

A component repository contains several implementations of components that have the functionality of data compression, specified with the provided property compression. They are diferentiated through the values of their non-functional attributes. Let us consider two different components, C1 and C2, specified as follows:

Component C1:

Property compression with attributes

compression_rate =crisp(0.6)

memory_consumption=crisp(512)

speed=fuzzy(slow)

Component C2:

Property compression with attributes

compression_rate =crisp(0.8)

memory_consumption=crisp(1024)

speed=fuzzy(medium)

Each of the three attributes is defined as a linguistic variable with three terms, as follows:

domain(compression_rate) = {low, medium, high}

domain(memory_consumption) = {low, medium, high}

domain(speed) = {slow, medium, fast}

For each linguistic variable, the parameters a1, a2, a3 defining the shape of the membership functions are defined. In our example, in case of the attribute compression_rate, these values are defined as a1=0.3, a2=0.6, a3=0.75.

It is important to note that a linguistic variable that characterizes an attribute can have different meanings in the context of different properties. The domain and the shape of a linguistic variable can be redefined in the context of different properties. For example, the attribute memory_consumption can be attached to very different functional properties: a compression component, a sorting component, a signal processing component, etc. In each of these cases, there are different expectations about the amount of memory needed: an amount of memory that is considered normal (medium) for signal processing is considered very high if it is required by a compression component. Thus the values a1, a2, a3 and maybe also the number of the linguistic terms will be defined different in the context of every property.

3.2 Generation of Fuzzy Rules from Requirements

3.2.1 Principle of Fuzzy Rule Generation

A new property matching mechanism is defined.

In general, a requirement as:

Req property P with attributes A1=V1 and A2=V2 and … An=Vn

is handled in the following manner:

First, the basic functionality is ensured, matching properties names according to the classical composition strategy. Usually several solutions result from thist first step.

Second, the preliminary solutions are selected and hierarchized according to the degree of attribute matching. This is done by fuzzy logic. The given requirement is translated into the corresponding rule:

If A1=V1 and A2=V2 and … An=Vn then decision=select

The generation of the fuzzy rules is done automatically starting from the requirements.

Very often, the required attributes are not values, but rather are required to be at least (or at most) a given value, A>=V or A<=V. For example, the speed is usually not required to be medium, but at least medium.

In general, a requirement containing the attribute expression A>=V will be translated into a set of i rules, for all Vi>=V:

If A=Vi then decision=select

3.2.2 Automatic Extension of the Fuzzy Rules Set

Several rules are generated from one requirement. In order to relax the selection, it is considered a match even if one of the linguistic variables in the premises matches only a neighbor of the requested value (the predecessor or the succesor). In this case the decision of selection is a weak one. In the case that more than one linguistic variable in the premise matches only neighbor values (while the rest match the requested fuzzy terms), the decision is a weak reject. In the extreme case that all linguistic variables in the premises match neighbor values, the decision is a weak reject. In all the other cases, the decision is a strong reject.

For example, in the case of a requirement containing two attributes, A1=V1 and A2=V2, the complete set of generated rules is:

The dirrectly generated rule is:

If A1=V1 and A2=V2 then decision=strong_select

The rules generated if one of the linguistic variables in the premises matches only a neighbor of the requested value are (maximum 4 rules):

If A1=pred(V1) and A2=V2 then decision=weak_select

If A1=succ(V1) and A2=V2 then decision=weak_select

If A1=V1 and A2=pred(V2) then decision=weak_select

If A1=V1 and A2=succ(V2) then decision=weak_select

In this case there are a maximum number of four generated rules, if neither V1 nor V2 are extreme values of their domains. If a value is the first value in the domain it has no predecessor, if it is the last value in the domain it has no successor.

The rules generated if more then one of the linguistic variables in the premises matches only a neighbor of the requested value are (maximum 4 rules):

If A1=pred(V1) and A2=pred(V2) then decision=weak_rej

If A1=succ(V1) and A2=pred(V2) then decision=weak_rej

If A1=pred(V1) and A2=succ(V2) then decision=weak_rej

If A1=succ(V1) and A2=succ(V2) then decision=weak_rej

For all the rest of possible combinations of values of A1 and A2 the decision is strong reject. The number of rules from this category depends on the number of terms of the linguistic variables A1 and A2.

3.2.3 Specifying the Relative Importance of Attributes

In order to allow the user to specify which attributes are more important and to treat them accordingly, different weights can be declared in the requirement specification. These weights describe how strict is a certain requirement: very strict, strict, normal, or less.

A requirement will be expressed for example in the following manner:

Req property P with attributes

A1=V1/imp=strict and A2=V2/imp=normal and … An=Vn/imp=less

The process of automatic generation of the extended set of fuzzy rules, presented in the previous paragraph considering the case when all attributes have normal importance, is adapted to deal with the attributes of different importance.

The attributes of strict importance will never be replaced by neighbor values in the generated fuzzy rules. The attributes of less importance will be replaced by neighbor and also second neighbor values in the generated rules.

3.2.4 Example

An application requires property compression with attributes having the values compression_rate=*medium* and speed=*medium*:

Req property compression with attributes

compression_rate=medium and speed=medium

No relative importances are specified for these two attributes, thus they are considered of equal importance.

The first rule which is generated is the direct rule:

[R1] If compression_rate=medium and speed=medium then

decision=strong_select

As illustrated in paragraph 3.1.2, the domains for compression_rate and speed contain each three linguistic terms: *low*, *medium*, *high* for compression_rate and *slow*, *medium*, *fast* for speed. For compression_rate, pred(*medium*)=*low* and succ(*medium*)=*high* while for speed pred(*medium*)=*slow* and succ(*medium*)=*fast*.

The rules generated for one different neighbor are:

[R2] If compression_rate=low and speed=medium then

decision=weak_select

[R3] If compression_rate=high and speed=medium then

decision=weak_select

[R4] If compression_rate=medium and speed=slow then

decision=weak_select

[R5] If compression_rate=medium and speed=fast then

decision=weak_select

The rules generated for two different neighbors are:

[R6] If compression_rate=low and speed=slow then

decision=weak_reject

[R7] If compression_rate=high and speed=slow then

decision=weak_reject

[R8] If compression_rate=low and speed=fast then

decision=weak_reject

[R9] If compression_rate=high and speed=fast then

decision=weak_reject

In this case, there are no rules where the decision is strong_reject, because both of the linguistic variables have only two terms and the requested term was the middle term.

We must also remark that the requirement was expressed using only the equality operator =. Thus, bigger values than the requested one are not considered to be best matches. If the user wants to consider better values than the requested one as best matches (for example, value *fast* for speed to be not considered worse that value *medium*) he must explicitly specify this in the requirement as speed >= medium. In this case, the rules that have speed=fast in the premise will have as conclusion a stronger selection decision.

3.3 Using Fuzzy Inference for Component Selection

3.3.1 The Principles

The selection of components corresponding to a set of requirements is done in several steps: First, the matching of properties according to the classic composition strategy resumed in paragraph 2.2 results in a list of potential solutions. Second, all the required attributes are used to generate the set of fuzzy rules as described in paragraph 3.2. Finally, the potential solutions obtained in the first step are hierarchized with help of fuzzy inference over the set of rules.

Given a fact A' and a rule $R_{A\rightarrow B}$, fuzzy inference means the composition A'$\circ$ $R_{A\rightarrow B}$ in order to obtain the conclusion B'= A'$\circ$ $R_{A\rightarrow B}$.

In our case, a fact A' is an expressions containing attributes of one candidate solution combined using logical operators, the premises A of the fuzzy rules are composed of attributes of the requirement, while the conclusion is the linguistic variable decision.

The premises of the rules are composed of several attributes, the degrees of activation of each premise are combined using the corresponding logical operators (AND, OR). The AND operator is implemented by *minimum* in fuzzy logic and the OR operator is implemented by *maximum* in fuzzy logic.

When more than one rule is active, the consequents of all active rules are combined through the union operator, which is implemented as a maximum between the membership functions of the partial conclusions. Most often, the result of the fuzzy inference has to be a crisp value, obtained by an averaging procedure applied on the partial conclusions, process that is called defuzzification. A widely used defuzzification method is the center of gravity.

The results obtained through defuzzification of the conclusions obtained for each candidate solution lead to their hierarchisation.

3.3.2 Example

Let us consider the following scenario to illustrate how the inference process works on selection of components. An application requires property compression with attributes values compression_rate=medium and speed=medium. From this requirement, the set of fuzzy rules illustrated in paragraph 3.2.4 are automatically generated. The component repository contains different component implementations for property compression, having different values for the nonfunctional attributes. Let suppose that there are available N different component implementations, two of them being these specified in the example of paragraph 3.1.2. For each of the N candidate components the value of the selection decision is calculated.

Figures 2-5 illustrate how each of the generated rules is composed with the fact represented by the specification of component C1 (with comp_rate=0.6 and speed=slow). Only four of the generated rules are activated, all other rules don't influence the inference process because their terms are not intersected by the facts resulting from the specification of component C1.

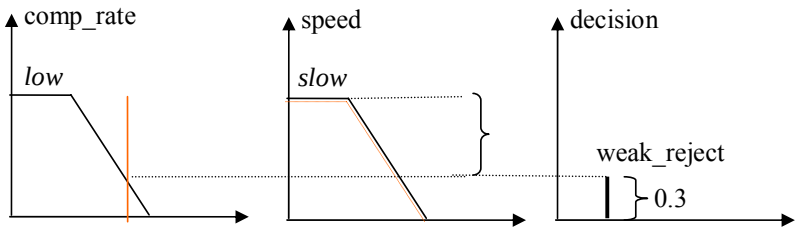

Figure 2

Rule: if compression_rate=*low* and speed=*slow* then decision=*weak_reject*.

Facts: comp_rate=0.6, speed=slow

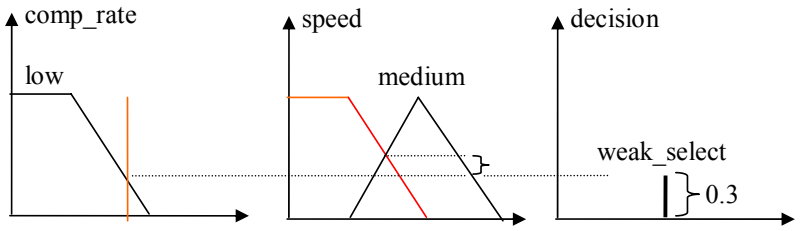

Figure 3

Rule: if compression_rate=*low* and speed=*medium* then decision=*weak_select*.

Facts: comp_rate=0.6, speed=slow

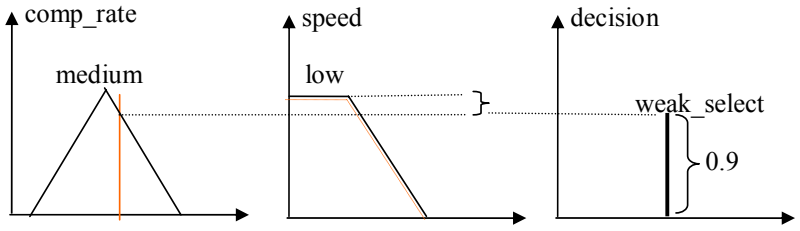

Figure 4

Rule: if compression_rate=*medium* and speed=*slow* then decision=weak_select.

Facts: comp_rate=0.6, speed=slow

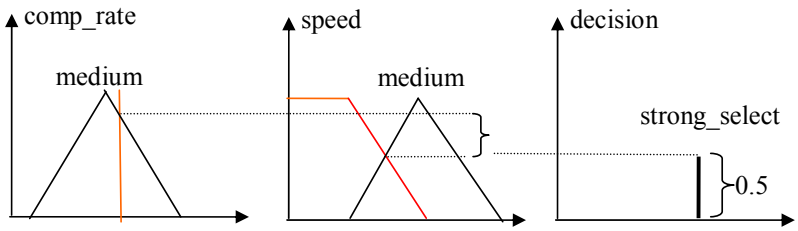

Figure 5

Rule: if compression_rate=*medium* and speed=*medium* then decision=*strong_select*.

Facts: comp_rate=0.6, speed=slow

The decision is the union between the partial conclusions weak_reject=0.3, weak_select=0.9, weak_select=0.3 and strong_select=0.5. Applying as a defuzzification method the the center of gravity for these partial conclusions, it results a decision value that classifies the component C1 as acceptable.

Similarily, fuzzy inference is also done for all the other candidate components. They can be ranked according to their decision values, the component with the decision value placed nearer to the right end of the domain (nearer to strong_select) being the best choice.

4 Related Work

Many different approaches have been proposed for the specification of components: interface description, formal specification methods, behavioural specification, architectural description. In previous work, we also developed a compositional model at the architectural level, the composable components model and its specification formalism, CCDL [8], [10]. A critical issue are the non-functional properties of components, as speed, memory consumption, usability, etc.

Since the use of fuzzy logic for the specification and retrieval of software components is a new approach, only a few experimental results are described in the literature. Two notable approaches are these of Zhang et al [12] and that of Cooper et al [2].

The work of Cooper et al [2] focuses on gathering specification data for individual components by tests and on the process of fuzzification of these specifications (their representation by membership functions). In their article they do not emphasis on the process of quering and decision making process for retrieving components that correspond to certain required properties, which instead is the main focus of our work. Another particularity of our work is that the individual

components do not have to bespecified with fuzzy properties. The advantage of this is that the fuzzyfication information (the shape of the membership functions) is described independent of individual components as a domain specific information.

Zhang et al [12] present a component matching system targeting automatic component searching and matching across the internet. The system is based on XML specifications and supports user queries that are specifications with incomplete/uncertain attributes. The matching approach used in their work is not fuzzy inference as it is used in our work. In [12], a query is parsed into a document object model (DOM) and the DOM is transformed to an internal tree-structured model. After this, fuzzy logic scoring and aggregation algorithms are applied to the internal tree structure to provide a ranked set of candidate approximate matches. In our approach, the matching is based directly on fuzzy logic inference which can be implemented in a generic manner.

Conclusions

In this article we introduced a new approach for the specification and retrieval of software components. This solution is based on fuzzy logic and extends our previous work on architectural level specification. The advantages of this approach are: a natural treatment for certain non-functional attributes that cannot be exactly evaluated and specified, and a relaxed matching of required/provided attributes that don't have to always be precise.

References

[1] Felix Bachman, Len Bass, C Buhman, S Comella-Dorda, F Long, J Robert, R Seacord, Kurt Wallnau: Technical concepts of component-based software engineering, Technical Report CMU/SEI-2000-TR-008, Carneggie Mellon Software Engineering Institute, 2000

[2] Kendra Cooper, Joao Cangusu, Rong Lin, Ganesan Sankaranarayanan, Ragouramane Soundararadjane, Eric Wong: An Empirical Study on the Specification and Selection of Components Using Fuzzy Logic, in Proceedings of 8th International Symposium on CBSE, St. Louis, USA, May 2005

[3] Xavier Franch: Systematic formulation of non-functional characteristics of software, in Proceedings of the 3rd IEEE International Conference on Requirements Engineering, Colorado Springs, USA, April 1998, pp. 174-181

[4] Murat Koyuncu, Adnan Yazici: A Fuzzy Knowledge-Based System for Intelligent Retrieval, in IEEE Transactions on Fuzzy Systems, Vol. 13, No. 3, June 2005, pp. 317-330

[5] Hadef Mili, Estelle Ah-Ki, Robert Godin, Hamid Mcheick: An experiment in software component retrieval, in Information and Software Technology, Elsevier

[6] The Object Management Group: Catalog of Domain Specifications. http://www.omg.org/technology/documents/domain_spec_catalog.htm

[7] Mary Shaw: Truth vs knowledge – the difference between what a component does and what we know it does, in Proceedings of the 8th International Workshop on Software Specification and Design, pp. 181-185

[8] Ioana Sora, Pierre Verbaeten, Yolande Berbers: A Description Language for Composable Components, in Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science LNCS No. 2621, Springer, 2003, pp. 22-37

[9] Ioana Sora, Vladimir Cretu, Pierre Verbaeten, Yolande Berbers: Automating decisions in component composition based on propagation of requirements, in Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science LNCS No. 2984, Springer, 2004, pp. 374-388

[10] Ioana Sora, Vladimir Cretu, Pierre Verbaeten, Yolande Berbers: Managing Variability of Self-customizable Systems through Composable Components, in Software Process Improvement and Practice, Vol. 10, No. 1, Addison Wesley, January 2005

[11] Clemens Szyperski: Component Software: Beyond Object Oriented Programming, Addison Wesley, 2002

[12] Ting Zhang, Luca Benini, Giovanni De Micheli: Component Selection and Matching for IP-Based Design, in Proceedings of Conference on Design, Automation and Test in Europe (DATE), Munich, Germany, 2001, pp. 40-46

[13] H.-J. Zimmermann, Fuzzy sets theory – and its applications. Second, revised edition, Kluwer Academic Publishers, 1991