Automatic Matching of Software Component Requirements using Semi-formal Specifications and a CBSE Ontology

Andreas S. Andreou[1] and Efi Papatheocharous[2]
[1]*Department of Computer Engineering & Informatics, Cyprus University of Technology, Lemesos, Cyprus*
[2]*SICS Swedish ICT, Swedish Institute of Computer Science, Kista, Sweden*

Keywords: Components, Reuse, Semi-formal Specifications Matching, Ontology.

Abstract: One of the most significant tasks of component-based software development is concerned with finding suitable components for integration. This paper introduces a novel development framework that promotes reusability and focuses on assessing the suitability level of candidate components. A specifications profile is first created using a semi-formal natural language that describes the desired functional and non-functional properties of the component(s) sought. A parser automatically recognizes parts of the profile and translates them into instance values of a dedicated CBSE ontology, the latter addressing issues of components' reusability. Available components on the market are also stored as instances of the CBSE ontology. Matching between required and offered component properties takes place automatically at the level of the ontology items and a suitability ratio is calculated that suggests which components to consider for integration.

1 INTRODUCTION

Component-based software engineering (CBSE) is the scientific area involved with software development and reuse of existing components. According to Szyperski (2002), "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 is subject to composition by third-parties". This is particularly interesting in the real world as by increasing the possibility of reuse leads to more qualitative systems and reduces their time-to-market. The alliances formed between software creators, vendors and owners, however, depend heavily on the methods and techniques to support the development process. The most significant advantages of reusing existing software parts, either small units (functions, classes) or fully-fledged systems (COTS) may be summed up to the acceleration of the development process, the increased dependability of the reused software and the reduction of the associated process risk. Mili et al. (2002) define software reuse as "the process whereby an organization defines a set of systematic operating procedures to specify, produce, classify, retrieve and adapt software artefacts for the purpose of using them in its development activities." Although the software components industry is steadily growing with multiple brokers to serve reusers (followed up tightly with open source communities as well), there is still great need for methodological approaches to improve and automate the processes of searching, retrieving and analysing candidate components for integration. To this end, the present paper proposes a new component reusability framework, which focuses on the identification of components and their assessment in terms of required features (functional or non-functional) that demonstrates their suitability for integration according to a prescribed (or desired) profile. The main novel aspects of this approach consist of: (i) an EBNF-based profiling of the components, which describes desired properties of the components sought, and (ii) an automatic search and retrieval mechanism. The latter utilizes the profiling scheme and without human intervention it delivers the most suitable components in three simple steps: parsing the ontology profiles of the requested and available components, executing the matching algorithm and recommending (retrieving) the closest matches. To the best of our knowledge existing approaches do not offer such automated

S. Andreou A. and Papatheocharous E..
Automatic Matching of Software Component Requirements using Semi-formal Specifications and a CBSE Ontology.
DOI: 10.5220/0005378301180128
In *Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2015), pages 118-128
ISBN: 978-989-758-100-7

management of components' reuse processes.

The rest of the paper is organized as follows: Section 2 provides a brief literature review on the subject. The proposed approach for profiling and matching components is described in section 3. The section starts with an overview of the reusability framework, continues with a presentation of the semi-formal description of components specifications and ends with the presentation of the details of the matching process, including the dedicated CBSE ontology and the matching algorithm. Section 4 describes a preliminary experimental investigation and reports some interesting findings on the assessment of the proposed approach. Finally, section 5 concludes the paper and suggests further research steps.

2 LITERATURE OVERVIEW

The relevant component search and retrieval literature is rich with studies about COTS, while Quality of Service (QoS) is one of the most frequently used mechanisms for component matching. In addition, ontologies have offered common ground to the CBSE process, either for describing metrics or properties for assessing components, or supporting their matching process. A brief outline of some of those studies follows.

Zaremski and Wing (1997) were among the first to use formal specifications to describe the behavior of software components and to determine whether two components match. Chung and Cooper (2004) presented an approach that supports iterative matching, ranking and selection of COTS represented as sets of functional and non-functional requirements. The work of Iribarne et al. (2002) presented an extension of approaches dealing with component search and service matching in which components offer several interfaces. More specifically, they addressed service gaps and overlaps extending the traditional compatibility and substitutability operators to deal with components that support multiple interfaces. Yessad and Boufaida (2011) proposed a Quality of Service (QoS) ontology for describing software components and used this ontology to semantically select relevant components based on the QoS specified by the developer. Pahl (2007) presented an approach for component matching by encoding transitional reasoning about safety and liveness properties into description logic and a Web standards compliant ontology framework. Yan et al. (2010) attempted to address the lack of semantic description ability in component searching and retrieval by introducing a conceptual ontology and a domain ontology. The authors represented a component ontology library by a conceptual and a component graph. During the retrieval process, the retrieval pattern graph was matched with the component graph using a component retrieval algorithm based on graph patterns. Kluge et al. (2008) suggested an approach for matching functional business requirements to standard application software packages via ontologies. Seedorf and Schader (2011) introduced an enterprise software component ontology to establish a common understanding of enterprise software components, i.e., their types and relationships to entities in the business domain. Alnusair and Zhao (2010) proposed a semantic-based approach for retrieving relevant components from a reuse repository utilizing an ontology model comprising of three axes, source-code, component, and domain-specific ontology. Their experiments suggested that only pure semantic search that exploits domain knowledge tends to improve precision.

Althought it is evident that matching of component specifications through the use of ontologies is not new, the above studies show that it is also promising and worth pursuing. The above studies, however, do not cover adequately the practical perspective of component reusability as they: (i) either express component services in abstract ontology forms and/or provide matching algorithm descriptions sometimes with and other times without the use of ontology information, (ii) do not provide concrete yet simple descriptors of the component properties, which may be reused by tools or methods that could further aid the reuse process. The present paper aspires to fill this gap by introducing an integrated framework for components' reuse, which offers a layered approach that guides the reuse process. The first layer of the framework is the key component to the process as it is responsible to profile component specifications using an expressive and easily understood (by reusers and component developers) semi-formal natural language structure, able to capture properties useful for components' matching. This profile is then transformed into a more formalized ontological representation and a simple, yet efficient way, to use this representation for automatically matching components, based on the suitability level of candidate components calculated by comparing ontology tree instances.

3 AUTOMATIC MATCHING OF COMPONENT SPECIFICATIONS

3.1 Reusability Framework Overview

The proposed framework is depicted in Figure 1 and consists of five layers (sub-systems), each supporting a part of the component-based software development process as follows: (i) The *Description* layer is responsible for creating a profile which includes relevant information that describe the component(s) sought or offered. A stakeholder (reuser or component developer/vendor) defines the functional and non-functional requirements that must be fulfilled or that are offered depending on the role. The former essentially provides the anticipated or desired properties in terms of functionality, performance, availability, reliability, robustness etc., and the latter sketches the functional behaviour of the ready-made software part. (ii) The *Location* layer offers the means to search, locate and retrieve the component(s) of interest that match the profile. (iii) The *Analysis* layer provides the tools to evaluate the level of suitability of the candidate component(s) and yield matching results that will guide the selection of components for reuse. (iv) The *Recommendation* layer uses the information received from the profiling activities and produces suggestions to reusers as to which of the candidate component(s) may be best integrated and why, through a cost-benefit analysis. (vi) Finally, the *Build* layer essentially comprises a set of integration and customization tools for combining component(s) and build larger systems.

One of the challenges this framework proposes to addresses is related to narrowing down the component requirements for searching and locating appropriate components, considering a minimal set of criteria and associating the various candidates with a ratio value of suitability; the latter will enable reaching to a plan (or recommendation) on how to progress with a project, and how to integrate components into one fully-functioning system.

This work concentrates only on the collaboration of the *Description* and the *Analysis* layers, and describes a new way for automatic matching between desired and available components based on structured natural language and ontologies. Next, the steps of the process are provided: The first step involves describing the desired functional and non-functional properties of the component(s) sought in a specifications profile using a semi-formal natural language. In the second step, the profile is automatically parsed and certain textual parts are recognized, which are then translated into instance values of a dedicated CBSE ontology. This ontology is built so as to reflect various development issues from the components reusability aspects. The third and final step performs matching between required and offered components' properties, the latter being stored also as instances of the CBSE ontology. This matching takes place automatically at the level of ontology items and a suitability ratio is calculated that suggests which components to consider next for possible integration.

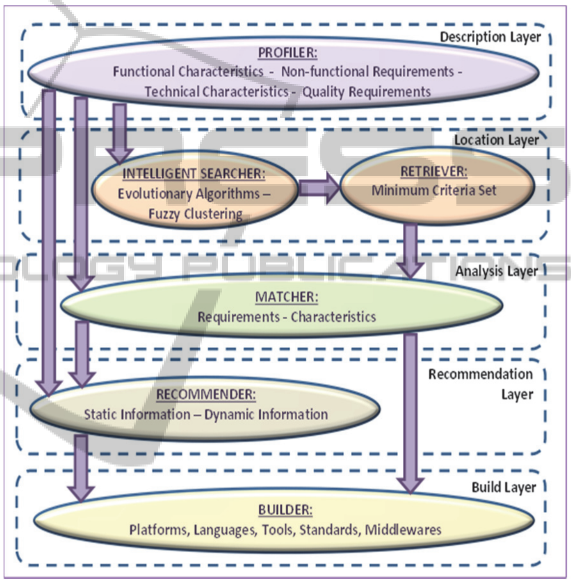
Description Layer
PROFILER:
Functional Characteristics - Non-functional Requirements - Technical Characteristics - Quality Requirements
Location Layer
INTELLIGENT SEARCHER:
Evolutionary Algorithms – Fuzzy Clustering
RETRIEVER:
Minimum Criteria Set
Analysis Layer
MATCHER:
Requirements - Characteristics
Recommendation Layer
RECOMMENDER:
Static Information – Dynamic Information
Build Layer
BUILDER:
Platforms, Languages, Tools, Standards, Middlewares

Figure 1: The layered architecture of the proposed reusability framework.

3.2 Components Profiling

Each component is profiled with information revolving around three axes, functional, non-functional and reusability properties, as follows:
<u>(i) Functional Properties:</u> One or more functional aspects included in the component are described here. More specifically, the services offered by the component are outlined, accompanied with the structure of the published interface (i.e., provides/ requires, detailing what services must be made available for the component to execute and what services are offered by the component during execution). Component contracts are also reported with the related Pre-conditions, Post-conditions, Invariants and Exception handling descriptions (i.e., cases where pre/post-conditions or invariants might be violated per method).
<u>(ii) Non-functional Properties:</u> Non-functional

constraints and quality properties are reported here. Performance indicators, resources requirements (e.g., memory and CPU) and other quality features (i.e., quality attributes based on the ISO 9126 standard, like availability (MTBF) and reliability).
(iii) Reusability Properties: It involves general information about the component which describes its context and way of use, its flexibility and other factors that are considered useful to reusers. Properties here include: application domain(s) for which the component is suitable, programming language(s), operating system(s) that is able to execute on, type of openness/extensibility (black, glass, grey, white), market price, some developer info (if available and reported for offered components), list of protocols and standards supported (e.g., JMS/Websphere, DDS/NDDS 4.0, CORBA/ACE TAO, POSIX, SNMP and .NET), as well as accompanying documentation (if any), like design, administration and user manuals and test cases documentation.

The component properties descriptions are written in the Extended Backus-Naur Form (EBNF). Expressing the component descriptions in the EBNF allows us to formally prove key properties, such as well-formedness and closure, and hence help validate the semantics. The proposed grammar has been developed with the Another Tool for Language Recognition (ANTLR) parser generator (http://www.antlr.org/). ANTLR is a parser and translator generator tool that allows language grammars' definition in an EBNF-like syntax.

Table 1 presents the EBNF description of a component. As previously mentioned, this description is used as a template from both the component vendor who provides information to increase its number of successful reuses and the interested reuser who tries to locate the most appropriate component for integration. There are some differences in the two cases, though, that are denoted with comments (text in green which starts and ends with the symbol '*') and refer mostly to information about contracts, developer details and documentation, which are not among the key information that reusers need to define when searching for components; they rather constitute peripheral information which is offered by the component developer/vendor, in the case that such information is (made) available.

While reading the profile from top to bottom, the reuser/developer will find the definitions used for the component items. The reuser/developer starts by filling-in this information with giving a name and selecting a list of (one or more) services the component has to offer. Each service is defined by a primary functionality type, a secondary informative type and thirdly, an optional description. Primary types include general functionality offered, like I/O, security and networking, while the secondary type explicitly express the kind of function it executes, like authentication, video streaming, audio processing etc. For example, a service could be [*Security*, *Login Authentication*]. If a service is sought for, then the reuser assigns a *Requirement* value, either *Constraint*, which means it is absolutely necessary and a candidate component is rejected if it does not offer it, or *Desired*, which simply adds points to the suitability value of a candidate component. Interfacing information comes next where each service is decomposed into the various methods that implement its logic; a method is analyzed to its constituent parts of *Pre-conditions*, *Post-conditions*, *Invariants* and *Exceptions* (if any). This piece of information can be provided by the component developer/vendor. Non-functional requirements or properties are defined next by the reuser and developer/vendor respectively, the former denoting what the search is for (and can be either defined as mandatory or desired), and the latter denoting what the component has to offer.

Finally, both the reuser and the component developer/vendor fill-in general information useful for reusability purposes (application domain, programming language, OS etc.) with the reuser again denoting the level to which a certain feature is required (defined as mandatory or optional). It should also be mentioned that certain features in the sought profile may be assigned to specific values along with a characterization as to whether this feature should be minimised (i.e. the value denotes an upper acceptable threshold) or maximised (i.e. the value denotes a lower acceptable threshold) in the suitable components offered. For example, if performance should be confined under 15 seconds, then next to the performance indicator the values (15, *minimise*) should be entered.

3.3 Automatic Components Matching

3.3.1 CBSE Ontology

A dedicated CBSE ontology is developed to reflect development issues based on the reusability of components. The ontology essentially addresses the same property axes and adheres to the same semantic rules of the component profile so that an automatic transformation of the latter to instances of

Table 1: Profile of a component in EBNF.

```
DIGIT ⇐ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ;    INTEGER ⇐ DIGIT {DIGIT};
CHAR ⇐ A | B | C | ... |W | a | b | c | ... | W | ! | @ | # | … ;    STRING ⇐ CHAR {CHAR} ;
Variable_type ⇐ CHAR | INTEGER | … ;    Variable_name ⇐ STRING
Primary_Type ⇐ ‘ Input ’ | ‘ Output ’ | ‘ Security ’ | ‘ Multimedia ’ | ‘ Networking ’ | ‘ GUI ’ | … ;
Secondary_Type ⇐ ‘ Authentication ’ | ‘ Data processing ’ | ‘ Video ’ |  ‘ Audio ’ | ‘ File access ’ | ‘ Printing ’ | … ;
Details_Description ⇐ CHAR { CHAR } ;
Min_Max_Type ⇐ ‘ Minimise ’ | ‘Maximise’ |
Required_Type ⇐ ‘ CONSTRAINT ’ | ‘ DESIRED ’ |
Service ⇐ ‘ S ’ INTEGER Primary_Type, Secondary_Type { Details_Description } Required_Type ;
Service_List  ⇐ Service { Service }
Operator ⇐ ‘ exists ’ | ‘ implies ’ | ‘ equals ’ | ‘ greater than ’ | ‘ less than ’ |…
Condition ⇐ Variable_Name Operator { Value } { Variable }
Precondition ⇐ Condition { Condition }; (* IF THESE ARE PROVIDED BY DEVELOPER/VENDOR *)
Postcondition ⇐ Condition { Condition }; (* IF THESE ARE PROVIDED BY DEVELOPER/VENDOR *)
Invariants ⇐ Condition { Condition }; (* IF THESE ARE PROVIDED BY DEVELOPER/ DEVELOPER/VENDOR *)
Exceptions ⇐ Condition { Details_Description } { Exceptions }; (* IF THESE ARE PROVIDED BY
DEVELOPER/VENDOR *)
Method ⇐ ‘ M ’ INTEGER { Variable Variable_Type } { Precondition }  { Postcondition } { Invariant } { Exception }
; (* IF THESE ARE PROVIDED BY COMPONENT DEVELOPER/VENDOR *)
Service_analysis ⇐ ‘ Service ’ INTEGER ‘ : ’  ‘ Method ’ INTEGER ‘ : ’ STRING Method { Method } ;
Performance_indicators ⇐ [‘ Response time ’ (INTEGER) Min_Max_Type Required_Type | ‘ Concurrent users ’
                    (INTEGER) Min_Max_Type Required_Type | ‘ Records accessed ’ (INTEGER) Min_Max_Type
                    Required_Type | … ] { Performance_indicators } ;
Resource_requirements ⇐ [ ‘ memory utilization ’  (INTEGER) Min_Max_Type  Required_Type | ‘ CPU reqs ’
(INTEGER)  Min_Max_Type Required_Type | … ] { Resource_requirements } ;
                    Quality_features  ⇐ [ ‘ Availability ’  (INTEGER) Min_Max_Type  Required_Type | ‘ Reliability ’
                    (INTEGER) Min_Max_Type  Required_Type | … ] { Quality_features }
Application_domain ⇐ ‘ Medical ’ Required_Type | ‘ Financial ’ Required_Type | ‘ Business ’ Required_Type | …
{Application_domain} ;
                    Programming_language ⇐ ‘ C ’  Required_Type | ‘ C++ ’ Required_Type | ‘ Java ’ Required_Type
                    | ‘ VB ’ Required_Type | … ; { Programming_language}
Operating_systems ⇐ ‘ Windows ’ Required_Type | ‘ Linux ’ Required_Type | ‘ Unix ’ Required_Type | ‘ IOS ’
                    Required_Type | ‘ Android’  Required_Type | …  { Operating_systems } ;
Openness ⇐ ‘ black ’ Required_Type | ‘ glass ’ Required_Type | ‘ grey ’ Required_Type | ‘ white ’ Required_Type;
Price ⇐ INTEGER ;
Development_info ⇐ STRING; Developer ⇐ STRING; Version ⇐ STRING; (* IF THESE ARE PROVIDED BY
DEVELOPER/VENDOR *)
Protocols_Standards ⇐ [ ‘ JMS/Websphere ’ Required_Type | ‘ DDS/NDDS ’ Required_Type | ‘ CORBA/ACE TAO ’
                    Required_Type | ‘ POSIX ’ Required_Type | ‘ SNMP ’ Required_Type |…  { Protocols_Standards
                    }];
Documentation ⇐ [ ‘ Manuals ’ Required_Type | ‘ Test cases ’ Required_Type | … ] ; (* IF THESE ARE PROVIDED
BY DEVELOPER/VENDOR *)

SPECIFICATIONS PROFILE :
      ‘Specifications Profile ’ STRING ;   ‘Descriptive title ’ STRING ;
      ‘Functional Properties :’ Service_List ;
      ‘Interfacing :’ Service_analysis { Service_analysis };
      ‘Non-functional Properties :’ Performance_indicators  Resource_requirements  Quality_features ;
      ‘Reusability Properties :’ Application_domain  Programming_language  Operating_systems  Openness  Price
                                 Protocols_Standards  Documentation ;
```

the ontology is feasible. Figure 2 depicts the largest part of the ontology; some details have been intentionally left out due to figure size and space limitations. A component is fully described by instances of the ontology items and can therefore be used as the basis for the matching process that is described next. This process works at the level of the ontology tree rather than the textual descriptions of the profile as comparison between required and available components is easier and more profound, both computationally and graphically (visually).

3.3.2 Matching Process

Some researchers focus on components retrieval issues and propose different methods for description processing, like simple string (e.g. Mili et al., 1994), signature matching (e.g. Zaremsky and Wing, 1993) and behavioural matching (e.g. Zaremsky and Wing, 1997). The proposed approach may be considered as a hybrid method comprising of string and behavioural matching but in a different manner than the aforementioned studies. More specifically, the cornerstone of the matching process is a dedicated parser which identifies parts of the profile (functional and non-functional behaviour, interfaces, performance indicators, etc.) and translates them into the CBSE ontology. The parser first checks whether the profile is properly described in the context and semantics of the structure (presented in Table 1 using the ANTLR framework). Once successful, the parser proceeds with recognizing the parts of the profile and building the ontology tree of instances following the algorithm presented in Figure 3. The parser essentially builds ontology tree instances which describe the requested and the available components. The next step is the matching of properties between ontology items. The tree instance of the required component is projected on top of all other candidates assessing the level of requirements' fulfilment in multiple stages. The first stage requires that all constraints are satisfied. In this case, the list of services sought must be at least a subset of the services offered. The second stage, executed once all constraints are satisfied, calculates the level of suitability of each candidate component. A demonstration example for this stage is given in the experimental section.

A requested component P_r defines in its profile a set of constraints K that must be satisfied including number and type of services, performance and quality factors, resource requirements, protocols/standards and documentation. The matching between the discrete items in the profile of P_r and those of a candidate component P_c is determined through the following rules:

(A) P_c is a *suitable* candidate for P_r if and only if every item $k \in K$ is satisfied by the corresponding item in P_c. We denote this by $P_c \equiv cand\ P_r$

(B) P_c is an exact match of P_r if and only if every item l defined in P_r is offered by P_c. We denote this by $P_c \equiv P_r$.

It is clear that rule (B) subsumes rule (A). The level of suitability is calculated for each *suitable* candidate as the ratio of matched profile items required (i.e. that are actually offered by the candidate component) to the total items outlined in P_r. More specifically, a dissimilarity value is calculated which indicates, in case of multiple suitable candidates, which one is closer to what has been requested.

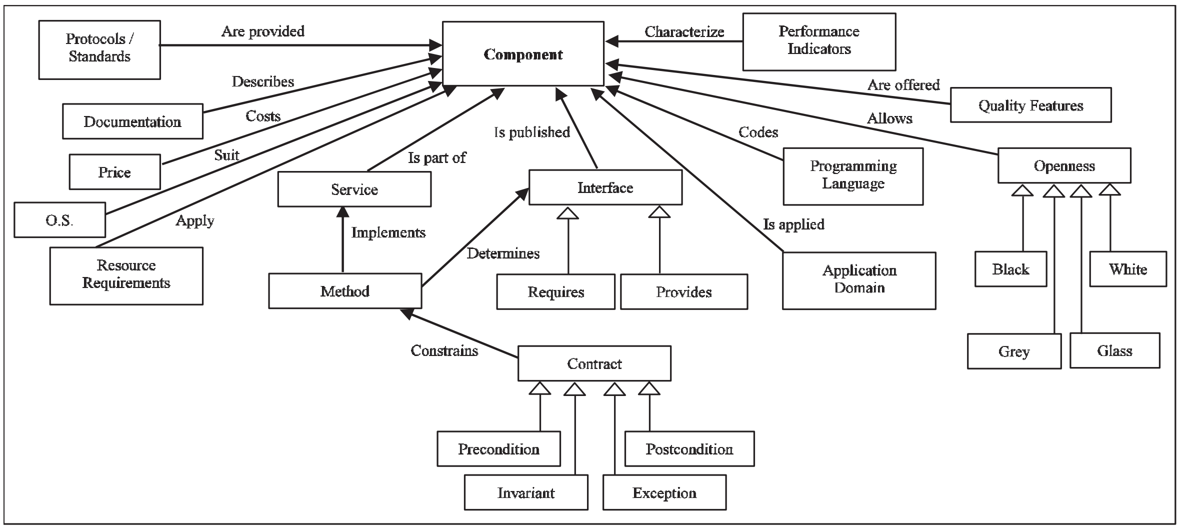

Figure 2: A CBSE ontology based on three axes, (i) Functionality, (ii) Non-functional aspects, (iii) Reusability properties.

```
Let Method(i)=set of methods implementing Service i
Trace 'Specification Profile' store STRING Name
Create Node Name
Start Parsing
  Trace 'Service List'
  Read N Services
  Trace 'Interfacing'
  For i=1 to N
  { Create Instance of Service i under node Name
    For each method j ∈ Method(i) do
     { Create Method j as node attached to Service i
       Determine Arguments A of Method j
       Create A as part of Interface node
       Determine Contracts for A and j's logic
        Create Preconditions, Post conditions, Invariants,
           Exceptions for j } }
  Trace 'Non-functional Properties'
  Read NOT_NULL non-functional properties
  For all NOT_NULL non-functional properties do
        Create Instances Performance indicators,
           Resource requirements, Quality features
  Trace 'Reusability Properties'
  Read NOT_NULL reusability properties
  For all NOT_NULL reusability properties do
        Create Instances Application domain,
           Programming language, Operating systems,
           Openness, Price, Protocols and Standards,
           Documentation
End Parsing
```

Figure 3: Algorithmic approach for the parsing process and ontology transformation.

We distinguish two types of properties, one of binary type (offered 'yes'/'no') and one of numerical type (e.g. price, response time). Matching properties of the former type presumes that all constraints are by default satisfied and its level is calculated simply by following the equations described hereafter: The binary dissimilarity is calculated as:

$$R_{bin} = \frac{1}{M}\sum_{i=1}^{M} \delta_{i,c,r} \qquad (1)$$

where

$$\delta_{i,c,r} = \begin{cases} 0\,, & \text{if property } i \text{ required in } P_r \text{ is offered by } P_c \\ 1\,, & \text{if property } i \text{ required in } P_r \text{ is not offered by } P_c \end{cases} \qquad (2)$$

and M the number of binary properties defined in P_r.

The numerical type is associated with minimum and maximum acceptable values. Therefore, matching of numerical properties is essentially another assessment of dissimilarity, which is performed by measuring how far from the optimal value (either maximum or minimum) lies the offered property value. We distinguish two cases:

(i) The property is mandatory (constraint). The candidates in this case satisfy the lower or upper bound of the defined feature value. Therefore, the distance between the values of the required and offered components is calculated by:

$$dC_{i,MAX} = \frac{max_{v_i} - v_i}{max_{v_i} - min_{v_i}} \qquad (3)$$

for feature value maximization, and

$$dC_{i,MIN} = \frac{v_i - min_{v_i}}{max_{v_i} - min_{v_i}} \qquad (4)$$

for minimization, while the total numerical dissimilarity for the constraints is calculated as:

$$R_{num,const} = \frac{1}{T}\sum_{i=1}^{T} dC_{i,\{MAX,MIN\}} \qquad (5)$$

(ii) The property is, not mandatory, but desired. In this case some of the values of the candidates satisfy the bounds defined in the desired components and some do not. Therefore, the distance between the desired property values v_d and the values offered by the candidate components v_i is calculated by:

$$dD_{i,MAX} = 1 + \frac{v_d - v_i}{max_{v_i,v_d}} \qquad (6)$$

$$dD_{i,MIN} = 1 - \frac{v_d - v_i}{max_{v_i,v_d}} \qquad (7)$$

for feature value maximization and minimization respectively. The total numerical dissimilarity for the desired features is then calculated as:

$$R_{num,des} = \frac{1}{M}\sum_{i=1}^{M} dD_{i,\{MAX,MIN\}}. \qquad (8)$$

In the above equations, max_{v_i,v_d} is the maximum value of the property between all candidates and the desired component, while T and M are the numbers of numerical properties that are mandatory and desired respectively.

The total value for the numerical properties is:

$$R_{num} = \frac{R_{num,const} + R_{num,des}}{2} \qquad (9)$$

The total dissimilarity value for a suitable candidate component is then calculated as:

$$R_{tot} = \frac{R_{bin} + R_{num}}{2} \qquad (10)$$

It is clear from the above that the closer the dissimilarity value to zero the better the suitability level of a component. The recommendation task ranks suitable components in ascending order of dissimilarity and suggests the top n candidates.

4 EXPERIMENTAL EVALUATION

A preliminary experimental process was conducted aiming at addressing the following three key questions regarding the proposed approach: (i) How easy and straightforward is it to locate appropriate components? (ii) How "complete" is the process? (iii) How accurate are the results (i.e. recommended components)? The word complete appears in quotes in the second question as completeness is not a property that may easily be quantified; nevertheless, for the purposes of this evaluation, we assume that completeness will denote the level to which the proposed process supports the profiling (and therefore the processing) of all possible sources of information describing component properties.

The experiments were carried out by 25 subjects, 20 of which were graduate (master) students at the Cyprus University of Technology and 5 were software practitioners. The students held an undergraduate degree in Computer Science and/or Engineering that included courses in Software Engineering (SE) and at the time of the experimentation they followed an advanced SE course with emphasis on CBSE and reusability. The practitioners consisted of software developers, 3 of which extensively make use of component reuse for the last 5 years and 2 produce components for internal reuse in their company for the last 3 years. All subjects underwent a short period of training (2 hours) on the proposed approach focusing mostly on the profiling scheme and the semi-formal structures of the natural language used. A total of 100 synthetic components were randomly generated with the help of the practitioners who inspected the elements produced and suggested corrections so as to correspond to realistic cases resembling real-world components. The components created were divided into 7 major categories: Login (10), Calendar (10), Address Book (10), Calculator (10), Task/Notes Manager (10), Clock (10) and GUI Widgets (Wallpapers (15), Window Style (15), Background/Fonts Style (10)). The multiple instances of the synthetic components for each category differed on attributes such as programming language, OS, openness, protocols/ standards and documentation, as well as on the performance indicators. The EBNF profile of each component was then created, followed by its transformation into an ontology instance of the component tree. Each subject was then asked to perform 10 different searches using a simple form (see Figure 4) where basically they inputted the desired functionality (primary, secondary), the values for certain performance indicators and their level of requirement (mandatory or desired). This information was also transformed in EBNF and then the ontology tree instance of the search item (component) was also created. Each search tree instance was then automatically matched against the available component instances in the repository. As this process is essentially an item-to-item matching of the tree instances, the classic metrics of precision and recall are not applicable here since the components retrieved were only those that satisfied all constraints for functionality and the rest of the features. Therefore, the candidate components returned were only the suitable ones which then competed on the basis of satisfying the rest of the properties sought for, calculating the level of suitability, as defined in eq.(10).

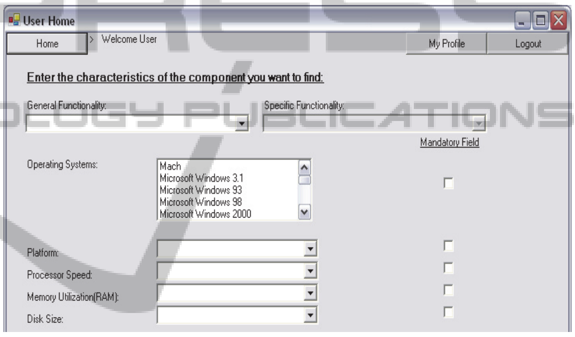

Figure 4: Excerpt of the component search form.

Table 2 shows part of the experimental process when searching for a Task Manager component, with functionality and features in the first column, preferences for the required component in the right most column and the five candidates in the columns in between. The lower part of this table lists the figures for the dissimilarity calculation described in eqs.(1)-(10). The figures clearly suggest that Component #2 is the candidate that best satisfies the search preferences, followed by Components #1, #4 and #5, that having similar characteristics to each other. This process was executed 10 times by each subject for each component category and the results were gathered and assessed qualitatively under the three questions described in the beginning of the present section related to ease of use, completeness and accuracy. At the end, the participants in the experimental study were asked to rate the approach on a five-point Likert scale ranging from 1-Very Low to 5-Very High for the focal point of each question.

The findings of the preliminary experimental results suggested the following: (i) The components

Table 2: Candidates' evaluation when seeking for a Task Manager component (C denotes constraint and D desired).

Task Manager	**1**	**2**	**3**	**4**	**5**	**SEARCH FOR**
Service Primary	input	input	input	input	input	Input (C)
Service Secondary	Data processing	Data processing	Data processing	Data processing	Data processing	Data processing (C)
Response Time (sec) (min)	10	12	***8***	***8***	9	12 (C)
Concurrent Users (max)	50	***100***	40	80	***100***	20 (C)
Memory utilization (KB) (min)	2	3	4	***1***	2	4 (C)
Total task supported (max)	200	1800	700	1900	***2000***	1500 (D)
Download history time (sec) (min)	6	8	22	***4***	20	18 (D)
Reliability (max)	90	***95***	92	93	90	90 (C)
Availability (max)	95	98	97	***99***	96	95 (C)
Application domain	ANY	ANY	ANY	ANY	ANY	ANY (C)
Programming language	C/C++	C/C++	Java	C/C++	.NET	C/C++ (D)
Operating systems	Windows	Windows	Windows Android	Windows Linux	Windows	Windows (C)
Openness	white	white	black	grey	white	White (D)
Documentation	Manual, Test Cases, Code, Comments, Design doc	Code, Comments, Design doc	Manual, Test Cases	Manual, Test Cases	Manual, Test Cases, Code, Comments	Code (D), Comments (D), Design doc (D)
Evaluation						
R_{bin}	0	0	0,714286	0,571429	0,285714	
R_{num}	0,8244589	0,47316	0,811688	0,270996	0,59632	
R_{tot}	**0,4122294**	**0,23658**	**0,762987**	**0,421212**	**0,441017**	

retrieved by the proposed approach were found suitable and among the top alternatives for all cases. It was also observed that the components returned as best candidates did not always possess the optimal numerical values in the corresponding properties sought, that is, the best values for the specific features (i.e. lowest time performance); they rather exhibited a good balance between numerical properties and also presented good ratings for the binary properties. This is clear in Table 2 where the optimal numerical values offered by the suitable components are marked in boldface and italic; it is evident that #4 holds the majority of optimal numerical values, yet it is not among the top 2. (ii) All subjects agreed that the method was quite easy to follow once trained, with a median rating of 4 (High). Especially with the use of the dedicated supporting tool, as some of the subjects stated, after their first few searches they felt quite comfortable with the approach and faced no problems in using it. (iii) Completeness was the feature that raised some questioning. Initial values by students rated this aspect with 4 (High), while practitioners gave the value of 2 (Low). Practitioners claimed that the approach should follow the same metrics and properties met in Service Level Agreements (SLA) which tend to become standard in the software industry, like those suggested by Czajkowski et al. (2002) and Mili et al. (2003). As this category of users was extremely important, a round of discussion was conducted through which the open nature of the profile scheme for a component was soon recognized as being able to cover any possible features or properties a reuser may seek, as long as the structured form followed for describing components encompasses these items. Therefore, practitioners agreed that the approach offers great flexibility in this respect and rated again completeness giving a median value of 4 (High). (iv) The discussion mentioned in the previous point gave birth to a suggestion for a possible extension to the approach: A priority or weighting scheme should be supported for the properties so that the reuser is able to define those features considered as more significant and therefore the assessment of candidate components will take this significance into account too, along with the rest similarity factors.

5 CONCLUSIONS

This paper addressed the issue of automatically matching specifications between components. A new component reusability framework was briefly introduced with a focus on the activities for matching required and offered properties. The matching process starts with the production of a special form of natural-language-based profile written in EBNF. The profile describes functional and non-functional aspects of components, as well as general reusability properties. A dedicated parser walks through the profile, recognizes certain sections and elements, and then translates them into instances of a special form of component-based ontology developed to support the component specification matching activities. A reuser uses the profile to describe what he or she looks for in a component using the EBNF notation, the latter being highly descriptive, while it allows to formally prove key properties and validate the semantics. Available components need also to be described by their developers/vendors under the same profiling details. The transformation of the profiles to ontology trees enables comparison at the level of instances which is used to assess if hard constraints are violated (i.e., absolutely necessary properties required are not offered by candidates) and if not, to calculate a dissimilarity metric that dictates the level of appropriateness of components for possible integration. Preliminary experimental results suggested that the proposed approach is accurate and suitable for adoption in the everyday practice of software reuse.

This work described a new idea with ample room for extensions and enhancements. Therefore, future work will include several research steps, some of which are outlined here: First of all, a more thorough experimentation will be carried out to validate the applicability and efficacy of the proposed framework. To this end, a series of experiments will be conducted utilizing open source components. Second, the retrieval parts will be enhanced by optimization techniques (e.g., evolutionary algorithms) for automating the process of locating candidate components. Third, the suggestions made during the experimentation phase will be incorporated in the approach, such as the prioritisation of the properties, which will guide the assessment of suitable components. Fourth, several aspects of the proposed approach will be parameterized so as to enable use customization and adaptation (e.g., weighting scheme of the matching algorithm). Last but not least, the dedicated software tool that supports the whole framework will be extended with capabilities for EBNF editing and ANTLR parsing during the construction of component profiles, as well as, graphical representation and visual inspection/comparison of ontology tree instances.

ACKNOWLEDGEMENTS

The work is partially supported by a research grant for the ORION project (reference number 20140218) from The Knowledge Foundation in Sweden.

REFERENCES

Alnusair, A., Zhao, T., 2010. Component search and reuse: An ontology-based approach. In *Proceedings of the IEEE International Conference on Information Reuse and Integration* (Las Vegas, USA, August 4-6, 2010). IRI2010, 258-261.

Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S., 2002. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Job scheduling strategies for parallel processing,* 153-183, Springer Berlin Heidelberg.

Chung, L., Cooper, K., 2004. Matching, ranking, and selecting components: a COTS-aware requirements engineering and software architecting approach. In *Proceedings of the International Workshop on Models and Processes for the Evaluation of COTS Components at 26th International Conference on Software Engineering, (Edinburgh, Scotland, UK, May 23-28, 2004)*. ICSE, 41-44.

Iribarne, L., Troya, J.M., Vallecillo, A., 2002. Selecting software components with multiple interfaces. In *Proceedings of the 28th Euromicro Conference* (Dortmund, Germany, September 4-6, 2002). EUROMICRO'02, 26-32. IEEE Computer Society Press.

Kluge, R., Hering, T., Belter, R., Franczyk, B., 2008. An approach for matching functional business requirements to standard application software packages via ontology. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference* (Turku, Finland, July 28 - August 1, 2008). COMPSAC '08, 1017-1022. DOI= 10.1109/COMPSAC.2008.147.

Mili, H., Mili, A., Yacoub, S., Addy, E., 2002. *Reuse based software engineering: techniques, organization, and measurement*. Wiley-Blackwell.

Pahl, C., 2007. An ontology for software component matching. *Int. J. Softw. Tools Technol. Trans*. 9, 2, 169-178.

Seedorf, S., Schader, M., 2011. Towards an enterprise software component ontology. In *Proceedings of the 17th Americas Conference on Information Systems* (Detroit, Michigan, August 4-7, 2011) AMCIS.

Szyperski, C., 2002. *Component Software: beyond object-oriented programming*, 2nd ed., Addison Wesley.

Yan, W., Rousselot, F., Zanni-Merk, C., 2010. Component retrieval based on ontology and graph patterns matching. *Journal of Information & Computational Science*, 7, 4, 893-900.

Yessad, L., Boufaida, Z., 2011. A QoS ontology-based component selection. *International Journal on Soft Computing* (IJSC), Vol.2, No.3, August 2011, 16-30. DOI : 10.5121/ijsc.2011.2302.

Zaremski, A.M., Wing, J.M., 1997. Specifications matching of software components. ACM T Softw Eng Meth, 6, 4, October 1997, 333–369.

Zaremski, A. M., Wing, J.M., 1993. Signature matching: A key to reuse (Vol. 18, No. 5, pp. 182-190). ACM.

Mili, H., Ah-Ki, E., Godin, R., Mcheick, H., 2003. An experiment in software component retrieval. Information and Software Technology, 45(10), 633-649.

Mili, H., Radai, R., Weigang, W., ... Elzer, P., 1994. Practitioner and SoftClass: a comparative study of two software reuse research projects. Journal of Systems and Software, 25(2), 147-170.

Keller, A., Ludwig, H., 2003. The WSLA framework: Specifying and monitoring service level agreements for web services. Journal of Network and Systems Management, 11(1), 57-81.