Towards a CBSE Framework for Enhancing Software Reuse: Matching Component Properties Using Semi-Formal Specifications and Ontologies

Andreas S. Andreou[1] and Efi Papatheocharous[2]

[1] Department of Computer Engineering & Informatics, Cyprus University of Technology, Limassol, Cyprus
[2] SICS Swedish ICT, Swedish Institute of Computer Science, Kista, Sweden

andreas.andreou@cut.ac.cy, efi.papatheocharous@sics.se

Keywords: Software Engineering, Components, Reuse, Semi-Formal Specifications, Ontology.

Abstract. A novel Component-based Software Engineering (CBSE) framework is proposed in this work that focuses on enhancing the reuse process by offering support for locating appropriate components. The architecture of the framework comprises of five interrelated layers, namely Description, Location, Analysis, Recommendation and Build. The scope of this work is to describe in detail the first and third layers, and provide the means to evaluate the suitability of candidate software components for reuse. The overall aim is to facilitate components' profiling and offer efficient matching of system and software requirements to increase the reusability potential of components. A specifications profile is created for each component using a semi-formal natural language that describes certain properties. A dedicated parser recognizes parts of the profile and translates them into instance values of a dedicated CBSE ontology that encodes these properties. Matching is then performed at the level of ontology instances between the available components and the components required by the developer. The framework recommends components based on a suitability ratio that calculates their distances from the desired properties.

1 INTRODUCTION

Component-based software engineering (CBSE) has emerged during the last two decades as a recognizable approach within the software development process that relies on extensive reuse of existing components and has attracted considerable research attention (Vale et al., 2015). The most significant advantages of reusing existing software components parts instead of developing systems from scratch, either small-grained units (functions, classes) or large-grained fully-fledged systems (Components Off The Shelf, COTS), are typically the acceleration of the development process, the increased dependability of the reused software and the reduction of the associated process risks. However, these benefits have been explicitly reported in only 33% of the studies according to the systematic literature study of Vale et al. (2015). Hence, software development with and for reuse still suffers from certain weaknesses that hinder their full exploitation potential. In our opinion, one of the most challenging weaknesses is the lack of efficient mechanisms to assess the suitability of candidate components for reuse. This is exactly the problem dealt within this work.

According to Szyperski's definition in (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 relevant with the way systems are developed today, as a lot of investment is put in using or buying ready-made components, through large-grained software reuse. The resulting systems are more qualitative and their time-to-market is significantly reduced, while at the same time, cost savings can be realized.

In the domain of CBSE two primary types of roles are distinguished: the software developer that develops the component from scratch and the reuser-developer or consumer, that is, the developer that makes use and integrates the ready-made components to develop a new software system. There are several alliances and agreements that need to be formed between software creators, vendors and owners. Completing a software product on time, within budget and with the required quality depends heavily on these relations, as well as on the methods and/or techniques utilized to support the development process. 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."

Nowadays, although the software components industry is steadily growing, and many emerging concepts such as Software as a Service (SaaS), Open Source Software (OSS) components and Components Off The Shelf (COTS) become more and more common in the way software is developed, there is still lack of efficient support to components' management, storing and retrieval. Although there are multiple brokers that try to serve reusers or consumers, and related communities are formed (for example the large Open Source community), still nowadays the need for more appropriate tools is emphasized. Tool support is in a way unavoidable for CBSE to succeed (Vale et al., 2015). Even though many efforts are recognized to exist today, Barn and Brown (1998) talk about the development of new generation tools of appropriate methods and tools that will encourage reuse and wider CBSE practice.

In a recent systematic mapping study of the CBSE area it was identified that the majority of the literature studies address primarily new solutions' functionality, while interaction (e.g., between components, component compositions, reusability property) and quality concerns are the second and third most common research topics respectively (Vale et al., 2015). In addition, among the list of research gaps reported, two of them significantly motivate the present work: Gap 4 and Gap 5 (as mentioned in the study above). The first indicated that CBSE concepts are still not fully integrated in industry or into the overall development and maintenance processes. In addition, the latter, also indicated that further research is needed to investigate how functionality, methodology and management must be further developed for CBSE to work in the first place, how CBSE can work efficiently and how components may be assessed. These research gaps pin-pointed by Vale et al. (2015) indicate that there is still great need for methodological approaches to improve and automate the processes of modelling, searching, retrieving and analyzing candidate components for integration.

To this end, the present work proposes a new CBSE reusability framework as an extension of the work in Andreou and Papatheocharous (2015). The architecture of the framework (as originally presented in Andreou and Papatheocharous (2015)) comprises of five interrelated layers, namely Description, Location, Analysis, Recommendation and Build. 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 component developers and reusers) semi-formal natural language structure. The purpose is to make it possible to capture properties useful for components' matching in an intuitive manner. 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 that is calculated by comparing ontology tree instances. The matching process is carried out in the third layer and the recommended solutions (components) are provided to the fourth layer of the framework that yields detailed recommendations, as to which of the candidate components may be best integrated and why.

In this work we focus and extend the implementations of the first and third layers. The framework focuses on the identification of components and their assessment in terms of required features (functional or non-functional properties). It demonstrates their suitability for integration according to a prescribed (or desired) requirements profile. The main novel aspects of this, is that the CBSE reusability framework approach consists of: (i) profiling of the components using the Extended Backus-Naur Form, which describes the desired properties of the required components, and (ii) an automatic search and retrieval mechanism for finding appropriate components for reuse. The latter utilizes the profiling scheme and without human intervention it delivers the most suitable components in three sequential steps: parsing the ontology profiles of the requested and available components, executing the matching algorithm and recommending the best matches. To the best of our knowledge existing approaches in the relevant literature do not offer such properties of filling-in the gap of automation and management of components' CBSE and reuse processes, neither have proper support for managing non-functional properties. The latter is also mentioned as Gap 6 in the Vale et al. (2015) study.

The most significant differences between this work and that of Andreou and Papatehocharous (2015) may be summarized to the following:

(a) The profile used to describe the components is modified and enhanced so that both horizontal and vertical expansion is feasible. Horizontal expansion refers to adding new values for a fixed (pre-defined) property, while vertical means the ability to add new properties. In the latter case the parsing mechanism is modified accordingly to be able to recognize the extension.
(b) The matching process is extended and includes the option of assigning weights to certain properties. This addition makes it possible to increase the properties' significance and this is primarily used in the comparison between available candidates. Therefore, the overall suitability of components can be adjusted based on what reusers consider more important when looking for appropriate component-solutions.
(c) The recommendation layer is also slightly enhanced with information that reveals possible incompatibilities between the successful candidates and what the reuser tries to find. Such incompatibilities are mostly focused on differences between programming platforms used to develop the candidates, or operating systems supported, and would potentially result in increase of the time and effort required by the reuser to integrate the component with the rest of the system. This case only applies if the properties that present incompatibility have been declared as desired and not as constraints.
(d) The experimental process was significantly extended and now includes a second, larger stage of experiments with increased levels of complexity and size as regards the targeted software application that is to be developed through reuse activities, thus touching also upon issues of scalability and efficiency.

The rest of the chapter is organized as follows: Section 2 provides a brief literature review on the subject. The proposed approach CBSE reusability framework 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 a dedicated CBSE ontology and the matching algorithm. Section 4 describes the experimental process which is divided into two stages and reports some interesting findings on the assessment of the proposed approach framework. Finally, section 5 concludes the chapter and suggests further research steps.

2 LITERATURE OVERVIEW

The literature overview of this section focuses on relevant publications on matching required properties and components. 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 promising common ground to the CBSE process, either for describing metrics or properties for assessing components, or for supporting in some way their matching process. A brief outline of some of those studies follows.

Zaremski and Wing (1997) were among the first, to the best of our knowledge, 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.

Although it is evident from the above studies that matching of component specifications through the use of ontologies is not a new concept, their results also show that it is a promising and worth pursuing research subject. Their results exhibit several improvements but also emphasize the need for expansions. What is more important to highlight, is the fact that current studies do not cover adequately practical aspects of component reusability. This is because they: (i) express component services in abstract ontology forms and/or provide matching algorithm descriptions sometimes with and other times without the use of ontology information, or (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. This chapter aspires to address this need by introducing an integrated CBSE reusability framework for components' reuse, which offers a layered approach that guides the reuse process. The details of the framework are presented next.

3 CBSE REUSABILITY FRAMEWORK

3.1 Overview

The proposed CBSE reusability framework is depicted in Figure 1 and consists of five layers (sub-systems). Each layer supports a part of the CBSE development process as follows: (i) The Description layer is responsible for creating the component profiles, which includes the properties for the components offered or required. A developer (either component developer or reuser) defines the functional and non-functional requirements that are provided by or required from existing components depending on the role the developer has within the process. The former essentially provides the functional behavior and properties of the available components in terms of functionality, performance, availability, reliability, robustness etc., and the latter provides the required properties. (ii) The Location layer offers the means to search, locate and retrieve the components of interest that match the profile. (iii) The Analysis layer provides the tools to evaluate the level of suitability of the candidate components 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 components may be best integrated and why, through a cost-benefit analysis. (vi) The Build layer essentially comprises a set of integration and customization tools for combining components and building larger systems.

One of the challenges that the present work addresses is the issue of 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 can enable reaching to a plan (or recommendation) on how to progress with a project, and how to integrate components into one fully-functioning system. Therefore, in this work, we focus to describe only on the details of the activities carried out in the Description and the Analysis layers. We focus to describe the process for conducting automatic matching between required and offered properties of components based on a structured semi-formal natural language and using ontologies. The proposed matching process consists of the following three steps:

Step 1: The required functional and non-functional properties of the component(s) are first described in a specifications profile using a semi-formal natural language. Functional properties specify a function that a system or system component must be able to perform (ISO/IEC 24765-2010). Non-functional properties are software requirements that describe not what the software must do but how the software must do it (ISO/IEC 24765-2010). The standards (ISO/IEC) were used as inspiration on what kind of properties one might use to describe specifications. Details on the profile descriptions are given in subsection 3.2.

Step 2: In this step, the profile specified is automatically parsed and certain textual parts are recognized. These are then translated into instance values of a dedicated CBSE ontology (details of the CBSE ontology are described in subsection 3.3.1). This ontology is built so as to highlight various development issues from the perspective of components reusability.

Step 3: The 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. Details on the matching process are provided in subsection 3.3.2.

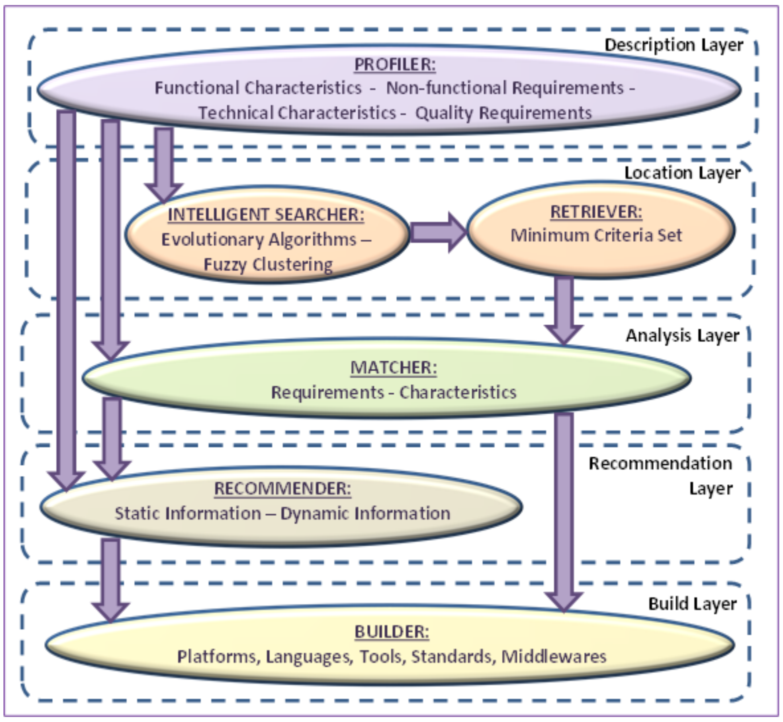
Description Layer
PROFILER:
Functional Characteristics - Non-functional Requirements - Technical Characteristics - Quality Requirements
Location Layer
INTELLIGENT SEARCHER:
Evolutionary Algorithms – Fuzzy Clustering
RETRIEVER:
Minimum Criteria Set
Analysis Layer
MATCHER:
Requirements - Characteristics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

Fig. 1. Layered architecture of the proposed CBSE reusability framework

3.2 Level 1: Description layer

Nowadays, the metrics and properties met in Service Level Agreements (SLA) tend to become standard in the software industry, especially for applications executing on distributed systems and the Cloud. The same concepts may easily be applied in our case where we target at providing a profiling scheme able to capture the properties of components for the purpose of reusing them in building larger applications. In this context, there are various approaches to SLA metrics, like those suggested by Czajkowski et al. (2002), Emeakaroha et al. (2010) and Paschke and Schnappinger-Gerull (2006). These studies make some useful categorization either in the context of performance metrics from both the hardware perspective (e.g., availability, failure frequency, processor time, etc.) and the software perspective (e.g., service times, response times, solution times, etc.), or from the point of view of the type of property described (e.g., time and scalar metrics). These categorizations have been carefully studied and certain concepts have been adopted and adapted in this work so as to reflect better the concepts of components description that are deemed necessary to support efficient reuse. The latter is realized by supporting effectively the process of finding the appropriate components for each case.

The first layer of the proposed CBSE reusability framework supports a specific type of component profiling, which uses information revolving around three axes: functional, non-functional and reusability properties. The selection of these axes was made targeting at describing a component from the perspective of the core functional aspects offered, the quality features and other constraints with direct effect on the functional properties, as well as a third viewpoint focusing on reusability issues. One may argue that the latter two types of properties may overlap. This is actually true as there is a thin line separating certain properties, while others may have the same meaning (e.g., use of standards). Nevertheless, we decided to differentiate between the two so as to emphasize on reuse issues and offer a way to handle information that may not involve general quality properties but at the same time is of particular importance to a component consumer, like for example those reported in the ISO-9126 standard. This becomes clearer in the description of the types of properties used to profile components that follows:

(i) Functional properties: One or more functional aspects included in the component are described. 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).

(ii) Non-functional properties: Non-functional constraints, such as quality properties are reported. 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) are described.

(iii) Reusability properties: The aspect of reusability of the component is described. It involves general information about the component related to its context, legacy and ways of current use, its flexibility and other factors that are considered useful to reusers. Properties here can include the application domain the component is developed for, the programming languages used, the operating system(s) that is able to execute on, the type of openness/extensibility (i.e., black, glass, grey, white), its price, developer information (i.e., details about the company or individual that created the component), a list of protocols and standards the component supports (e.g., JMS/Websphere, DDS/NDDS 4.0, CORBA/ACE TAO, POSIX, SNMP and .NET), as well as information about accompanying documentation (like design, administration and user manuals, and test cases). Some of the above properties even though important for the component' utilization, they are made optional in the implementation and thus it depends on the component developers to provide the corresponding information.

The component properties descriptions are written in the Extended Backus-Naur Form (EBNF). Expressing the component descriptions in the EBNF notations 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 developer and the reuser. The developer's motivation to provide this information in the best possible way is to increase the chances and frequency of successful reuses and the reuser needs to specify this information so as to be able to search and find the best possible alternatives for integration. One may observe that component properties descriptions will have to present some differences in the information provided from the two types of users. These differences are denoted in the comment lines (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 would need to define when searching for components, and they rather constitute peripheral component information from the reusability property descriptions. A key point here is the ease of extensibility of the profile presented in Table 1. We distinguish two types of expansion, horizontal and vertical. The horizontal expansion is realized by adding new values at the end of certain properties where applicable. This is indicated by the ellipsis (i.e., the punctuation mark consisting of three dots) after the last property value, for example after 'GUI' for Primary Type. The vertical expansion is similarly depicted in two specific parts of Table 1, namely the non-functional and the reusability properties, and it may be achieved by adding any new properties expressing them in the proper EBNF format and by modifying the parsing module so that it transforms the new property elements to the correct instances of the revised ontology scheme. This feature is not yet automatically supported; it is supported conceptually and it is left as part of our future work on the supporting software tool.

While reading the profile from top to bottom, the reuser finds the definitions used for the component items in the template. The reuser starts by filling-in this information, giving a name and selecting a list of (one or more) services the component is required 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 expresses 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. Non-functional requirements or properties are defined next by the reuser and developer (creator) 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 developer 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 minimized (i.e., the value denotes an upper acceptable threshold) or maximized (i.e., the value denotes a lower acceptable threshold) in the suitable components offered. For example, if response time should be confined under 15 seconds, then next to that performance indicator the values (15, *minimize*) should be entered.

Table 1. Profile of a component in EBNF form

```
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 ⇐ ‘ Minimize ’ | ‘Maximize’ |
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 *)

(* ====  INTERFACING  ======= *)
Service_analysis ⇐ ‘ Service ’ INTEGER ‘ : ’  ‘ Method ’ INTEGER ‘ : ’ STRING Method { Method } ;

(* ====  NON-FUNCTIONAL PROPERTIES  ======= *)
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 }
…
(* ====  END OF NON-FUNCTIONAL PROPERTIES; NEW ITEMS MAY BE ADDED HERE  ======= *)


(* ====  REUSABILITY PROPERTIES  ======= *)
Application_domain ⇐ ‘ Medical ’ Required_Type | ‘ Financial ’ Required_Type | ‘ Business ’ Required_Type | … {Appli-
          cation_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 ’ Re-
          quired_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 ’ Re-
                    quired_Type | ‘ POSIX ’ Required_Type | ‘ SNMP ’ Required_Type |…  { Protocols_Standards }];
Documentation ⇐ [ ‘ Manuals ’ Required_Type | ‘ Test cases ’ Required_Type | … ] ; (* IF THESE ARE PROVIDED BY
          DEVELOPER/VENDOR *)
```

```
…
(* ====  END OF REUSABILITY PROPERTIES; NEW ITEMS MAY BE ADDED HERE  ======= *)

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 ; …
```

3.3 Level 3: Analysis Layer

3.3.1 Dedicated 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 the ontology is feasible. Figure 2 depicts the largest part of the ontology; some details have been intentionally left out to make the graphical representation more readable. 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).

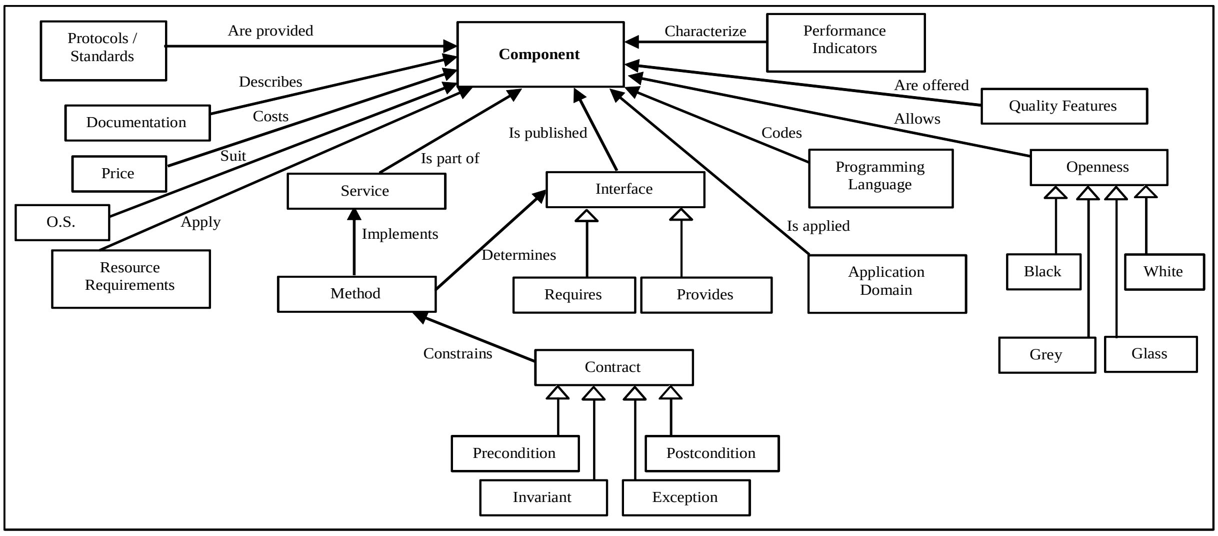

Fig. 2. The CBSE ontology based on three axes: (i) Functionality, (ii) Non-functional, and (iii) Reusability properties

3.3.2 Matching Process

The matching process defined in this work was inspired by considering the strengths and weaknesses of similar approaches identified in the relevant literature. 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 string and behavioral 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 behavior, 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 first part of 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:

Rule (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 $P_c \equiv cand\ P_r$

Rule (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 $P_c \equiv P_r$.

Clearly 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.

```
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
```

Fig. 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:

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

where

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

and *M* the number of binary properties defined in Pr.

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:

$$dC_{i,MAX} = \frac{max_{v_i} - v_i}{max_{v_i} - min_{v_i}} \tag{3}$$

for feature value maximization, and

$$dC_{i,MIN} = \frac{v_i - min_{v_i}}{max_{v_i} - min_{v_i}} \tag{4}$$

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

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

(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:

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

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

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

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

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:

$$R_{num} = \frac{R_{num,const}+R_{num,des}}{2} \tag{9}$$

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

$$R_{tot} = \frac{R_{bin}+R_{num}}{2} \tag{10}$$

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 series of experiments was designed and executed so that the usefulness, applicability, effectiveness and efficiency of the proposed CBSE reusability framework are assessed. Specifically, two sequential experimental stages were followed. The first stage comprised of searching and retrieving various components based on a set of properties, while the second involved investigating targeted reusability development activities and assessed in addition scalability. The second stage of experiments is considered a significant extension of the first, as it was considerably longer, in-depth and comprises of actual developments activities to implement a simple application with reuse and through the use of the proposed CBSE framework. Both experimental stages were used as feedback (based on the evaluation obtained) for the overall improvement of the proposed framework.

The experimental stages were carried out in total by 25 subjects, 20 of which were graduate (master level) students at the Cyprus University of Technology and the remaining 5 were software practitioners. The students hold 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 working in the industry, 3 of which extensively making use of component reuse for the last 5 years and 2 of which are responsible for producing components for internal reuse in their company for the last 3 years.

4.1 Experimental Stage A’

In the first experimental stage (A’), all subjects underwent a short period of training (approximately 2 hours) on the proposed CBSE reuse framework focusing mostly on the profiling scheme and the semi-formal structures of the natural language used. The target was to provide a first evaluation of the usefulness, applicability, effectiveness and efficiency of the framework.

The experiments conducted thus aimed at addressing the following four main questions regarding the proposed framework: (**Q1**) *How easy and straightforward is it for locating appropriate components?* The question mainly focuses on the ease of use, level of understandability and effective utilization of functional, non-functional and reusability properties to seek and locate candidate components sought for. (**Q2**) *How “complete” is the process for locating appropriate components?* Answering this question will essentially define whether there is enough information supported from the framework or if there exist key information that a reuser would like to search for and is yet not supported. The word complete appears in quotes as completeness is not a property that may easily be quantified. Nevertheless, for the purposes of this evaluation, we assume that completeness denotes the level to which the proposed process supports the profiling (and therefore the processing) of all possible sources of information describing component properties. (**Q3**) *How accurate are the results (i.e., the recommended components)?* This question refers to the quality of the results in terms of correctness and specifies the end-user's (the reuser in our case) satisfaction level. (**Q4**) *What is the efficiency of the process in terms of time/effort required by reusers to locate and retrieve the components they need?* This question refers to the quality of the results in terms of efficiency and effort required to spend and again relates to the reuser satisfaction levels.

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 and 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 (Wallworks (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).

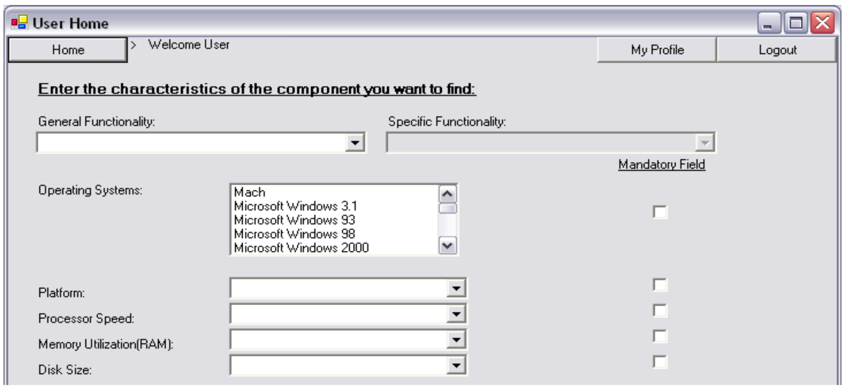

Fig. 4. Excerpt of the component search form

Table 2 shows details of the experimental process while one of the reusers was searching for a Task Manager component. Components' functionality and features appear in the first column, preferences for the required component in the right most column and the five (retrieved) 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 have 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 four main questions (Q1-Q4), described in the beginning of the present section, related to ease of use, completeness, accuracy and efficiency of the process and the results obtained. 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 to obtain the focal point of each question.

Table 1. 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**	

The findings of the experimental results (Stage A') suggested the following with respect to the questions:

(**Q1**) *How easy and straightforward is it for locating appropriate components?* All subjects agreed that the method was quite easy to follow once trained, with a median rating of 4 (High). The training was mentioned that it was not too difficult to go through and in terms of effort required it was acceptable. It is even easier to carry out the search 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.

(**Q2**) *How "complete" is the process for locating appropriate components?* Completeness was the feature that brought to light some questioning. Initial values by students rated this aspect with 4 (High), while practitioners gave the value of 2 (Low). Some follow-up questions in the interviews we conducted the difference was obvious where it originated from. Practitioners, as they are more experienced with the variability of components in the real-world, emphasized that the approach should be more flexible and allow for more metrics and properties to take part in the profiling of a component. After a round of discussions, through which the open nature of the profile scheme for a component was explained and exemplified, and emphasis was given on how new properties may be inserted to satisfy further needs, practitioners were asked to rate again the question. They recognized that extensibility was just a matter of profile design and thus the current one could be extended. The approach is 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).

(**Q3**) *How accurate are the results (i.e., the recommended components)?* The components 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 Component #4 holds the majority of optimal numerical values, yet it is not among the top 2.

(**Q4**) *What is the efficiency of the process in terms of time/effort required by reusers to locate and retrieve the components they need?* The total time and effort spent to locate the appropriate components was quite limited, rated as 2 (Low) being confined to the actions required to describe the properties of the components sought. Automation of the whole process for returning appropriate components back to the reuser was acknowledged to have a decisive positive effect on the total time that had to be devoted, as stated from the practitioners.

The results from the first experimental stage, and especially Q3, gave birth, to a possible modification/extension of the approach. This modification included an additional implementation and application of a priority or weighting scheme for the framework properties. With this scheme the reuser will be able to define the properties considered as more significant than the rest and therefore, change the assessment of candidate components to take this significance into account too, along with the rest aforementioned similarity factors. This modification is explained further in Experimental Stage B' (subsection 4.2).

4.2 Experimental Stage B'

With similar experimental design, the second experimental stage was conducted. During the second experimental stage (B'), the proposed CBSE reusability framework was used by the reusers with the target to implement a small-scale software application and assess in addition scalability.

The same group of subjects, were asked to make use of the proposed framework and locate appropriate components that can be reused. They were given the following instructions. Develop a software Web application that supports the following:

- Multiple users
- User authentication mechanisms
- Management of client records (insert, update, delete)
- Management of product records (insert, update, delete)
- Order placement for purchasing products (search and browse products, place in or remove from shopping cart products)
- Card payment processing
- Logistics support (issue invoices, audit trails).

The target of the second experimental stage was twofold: First, to expand the use of the framework to a larger and functionally demanding experimental setup. The setup is more realistic, includes complex development tasks and comes across to integration issues for CBSE (resulting from reusing of existing components, such as the issue of synthesis and compatibility of individual parts). Second, to assess scalability and efficiency of the approach when scaling up complexity, handling at the same time challenges raised from the previous experimental stage.

As such, to meet the first target the approach was slightly extended at the Recommendation level so as to include a software module that performs checks to detect profound issues affecting compatibility, like the programming language or the operating system, and suggest a different ranking of the appropriate components based on the expected extra costs (time/effort) arising from such issues. This kind of recommendations was provided only in cases where the possible incompatibilities were associated with properties that were not declared as constraints.

To meet the second target, the framework was extended with the addition of a new element that provides a mechanism that allows a reuser to define varying degrees of significance among the desired properties. A weighting scheme was therefore applied during the process of describing the properties of the component sought using a Likert scale of 3 values indicating 1-low, 2-medium and 3-high level of significance.

A pool of 350 synthetic components was created and made available through a server. Each functional characteristic was satisfied by at least 5 generated components with varying properties (e.g., resource requirements, performance indicators, programming language, etc.). The reuser was handed a randomly generated list of properties (constraints and desired ones) to guide the selection of components that were appropriate for each case and was left free to interact with the system to set their priorities as regards the significance of the component features and then locate the most suitable ones for the functional tasks described above. As with the previous stage, each reuser was given access to the server using a simple GUI for searching and retrieving components based on the concepts of the proposed approach, with search properties being defined using primarily pull down menus, drop-down lists and check-boxes (Figure 4). Then, these property definitions were automatically transformed to their EBNF counterparts and corresponding ontology notions with no extra effort or visibility on behalf of the reuser.

The experimental stage B' spanned approximately one week's time, by the end of which the experiences obtained were shared among all participants in a closed session and the framework was rated once again. The results of the experiment were assessed in a similar way to the previous one:

(**Q1**) *How easy and straightforward is it for locating appropriate components?* Subjects answered that even with performing a wider-scope task the approach is still quite easy and straightforward. They rated it with a median rating of 4 (High). A threat to the validation of this result is however the fact that subjects were not the first time to have worked with the tool. The large size of the application that required the search of multiple components and the increasing complexity of the development task, minimizes the effect of this threat.

(**Q2**) *How "complete" is the process for locating appropriate components?* The extension of the framework to accommodate new values for fixed (pre-defined) properties as well as new types of properties was received positively by the experiment participants. They gave a median rating of 4 (High). A comment received during the discussions was that even if the second experimental process the subjects undertook is more realistic to the real working conditions in terms of complexity, the type of developments they might face in the future would definitely contribute to the level of richness of the properties, and thus enhance further the overall completeness levels.

(**Q3**) *How accurate are the results (i.e., the recommended components)?* The accuracy of the returned components was rated with a median rating 2 (Low). This was due to the random nature of the generated components, as well as of the list of properties each participant was supplied with, i.e., there were cases where the properties sought were not perfectly aligned between them. For example, the use of a component for displaying the list of products had quite short response time than that of the component inserting a product in the shopping cart. Therefore, participants discussed for a more realistic generation of component properties within the framework. Nevertheless, everybody agreed that this "anomaly" did not violate the outcome of the experimental process as at this stage what was important was to assess the applicability of the approach to more complicated situations. Thus the scalability issue of the framework even if not evaluated explicitly, was considered promising. In addition, the accuracy potential of the results was slightly improved with the weighting scheme applied in this experimental stage. The weighting scheme enabled reusers to narrow down the list of matches, especially in cases where the candidate components were similar or very close to each other with respect to the properties set. In addition, the recommendation module worked fairly well in the majority of the cases, although some participants pointed out that its use was not always helpful. There were cases where the incompatibility of the programming platforms in which the components were built hindered their composition (or at least recommended them not to be integrated), but their combination was by far better than that of other, that appeared as more compatible ones. This observation, combined with the fact that if a wrapper could easily be developed to handle the incompatibility issues, led to the conclusion that the recommendation module should be further enhanced with more sophisticated ways of suggesting the use or not of specific types of components. Nevertheless, this was already a known "weakness" of the recommendation engine, as its purpose was just to demonstrate how its use may offer enhanced support to the selection of similar components, something which was acknowledged by all participants.

(**Q4**) *What is the efficiency of the process in terms of time/effort required by reusers to locate and retrieve the components they need?*

Overall, the efficiency was found again high (median value was 4), despite the fact that the search process was multifaceted compared to experimental stage A'. Indeed the results confirmed that the size of the application under development did not add significantly to the complexity of the way the approach was used, but rather affected the time for locating all component instances for each case as expected.

5 CONCLUSIONS

This work addressed a specific topic in the area of component based software engineering and more specifically the issue of automatic search and retrieval of software components by matching specifications. A new CBSE reusability framework was proposed which comprises different layers for describing, analyzing, locating and assessing the appropriateness of available components.

The work focused on the activities of the matching process between required and offered properties. This process initially produces a special form of natural-language-based profile written in EBNF, the latter being highly descriptive, while it allows formally proving key properties and validating the semantics. The profile describes three different categories of components' properties, that is, functional, non-functional and general reusability properties. A specially designed module parses 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. Using this profile developers/vendors of components offer details of what they have to offer to potential reusers who use the profile to describe what they look for using the same EBNF notation. The profiles are transformed into ontology trees, something that enables faster comparison between characteristics as this commences at the level of ontology instances. The matching process assesses if hard constraints are violated (i.e., absolutely necessary properties required are not offered by candidates) and if not, it calculates a dissimilarity metric that dictates the level of appropriateness of components for possible integration.

A two stage experimental process was followed, the first focusing on demonstrating and evaluating the applicability of the proposed approach, while the second further investigated efficiency and scalability issues through a more complicated reuse context. The experiments provided strong evidences that the proposed approach is accurate, complete and efficient, and therefore it may be regarded as suitable for adoption in the everyday practice of software reuse.

The framework introduced in this work may be conceived as a promising new idea with ample room for extensions and enhancements. Our 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. Finally, 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.

6 Acknowledgement

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

7 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.

Andreou, A.S., Papatheocharous, P., 2015. Automatic Matching of Software Component Requirements Using Semi-Formal Specifications and a CBSE Ontology. In Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software (Barcelona, Spain, April 29-30).

Barn, B., Brown, A.W, 1998. Methods and tools for component based development. In Proceedings of IEEE Technology of Object-Oriented Languages (TOOLS'98).

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.

Emeakaroha, V. C., Brandic, I., Maurer, M., Dustdar, S., 2010. Low level metrics to high level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In Proceedings of the International Conference on High Performance Computing and Simulation (HPCS) (48-54). IEEE.

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.

ISO/IEC/IEEE 24765:2010, 2010, Systems and software engineering - Vocabulary, ISO/IEC/IEEE 24765:2010.

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.

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

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

Mili, H., Radai, R., Weigang, W., Strickland, K., Boldyreff, C., Olsen, L., Witt, J., Heger, J., Scherr, 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.

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

Paschke, A., Schnappinger-Gerull, E., 2006. A Categorization Scheme for SLA Metrics. Service Oriented Electronic Commerce, 80, 25-40.

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.

Vale, T., Crnkovic, I., de Almeida, E.S., da Mota Silveira Neto P.A., Cerqueira Cavalcanti, Y., de Lemos Meira, S.R., 2016. Twenty-eight years of component-based software engineering, Journal of Systems and Software, 111, January 2016, 128-148, ISSN 0164-1212, http://dx.doi.org/10.1016/j.jss.2015.09.019.

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.