Semantic matching to achieve software component discovery and composition

Sofien Khemakhem, Khalil Drira, Mohamed Jmaiel

Semantic matching to achieve software component discovery and composition

Sofien KHEMAKHEM[1] , Khalil DRIRA[2,3] and Mohamed JMAIEL[1]
[1] University of Sfax, National School of Engineers,
Laboratory ReDCAD, P.B.W. 3038 Sfax, Tunisia
e-mail: Khemakhem_sofien@yahoo.fr, Mohamed.jmaiel@enis.rnu.tn
[2] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[3] Université de Toulouse, LAAS, F-31400 Toulouse, France
e-mail: Khalil@laas.fr

Abstract

In CBSE, current approaches based on software component are in short of flexibility because of lacking semantic. In this paper, we introduce an extended semantic discovery of software components in which non functional properties of components are considered. We also introduce an ontology supported automatic component composition method to realize the integration of the adequate component composite in the current project.

1 INTRODUCTION

Since the middle of 1990's, object-oriented programming becomes a key component in software engineering. Many reserach in software development focus on software reuse, (KDJ11) while the software composition becomes an important component in software engineering. To resolve the difficulty of manually building a composite component, there have been many studies on the automated composition of software components (SCLK06). Previous works specify the semantic description of a component using service specification languages such as OWL-S (CTJD13), WSMO (dBBD$^+$05), and SAWSDL (ABD$^+$07) based on a domain ontology, which is built by an ontological language such as OWL (SWM04). Likewise, a developer's request is specified in terms of inputs, outputs, preconditions and effects. From these specifications, they automatically build composite components using various AI (Artificial Intelligence) techniques. Previous methods construct composite components, which take the inputs entered by a developer and return the outputs requested, by repeatedly discovering and chaining appropriate components. When two components are chained, the preceding component should satisfy the precondition of the following component. Generally, two components with identical inputs, outputs, preconditions, and effects are regard as identical components. Therefore, if a composite software component satisfies the inputs, outputs, preconditions, and effects requested by a developer, it is regarded as satisfying the user requirement. However, when components or user requirements do not have pre-conditions and effects, the conventional methods may generate composite components, which differ from a developer's needs. In general, two components with

the same IOPE (input, output, pre-condition and effect) are considered to be identical each other (CF12). However, non-functional properties (JA10), which provide certain information about component constraint to users, may not need any preconditions and have no effects after execution. In previous works query is based on IOPE, cannot compose correctly if the non-functional aspect of the disred composite component is not specified. In several cases, the non-functional constraints play a decisive role in the choice of the most powerful composite component. previous works have exponential time complexity in proportion to the number of available components, since they have to consider every possible combination of available components. In this paper we aim to define a unified and a complete approach to ameliorate the reuse of software components in the CBD. Our approach encompasses the functional and the non-functional aspect in different stage: description, discovering, composition and integration. To achieve our objective we develop three ontologies. Two ontologies are used to discover atomic and composite components which are the discovery and the shared ontology. The third ontology is the integration ontology which describes the component's internal structure to facilitate its integration in the current work after its selection. To improve the integration process we also use, Output-Matching-Service and Input-Output-convertor are service types used in matching parameters. This paper is organized as follows: Section 2 presents the discovery and the integration ontology. We will devote section 3 to detail the shared ontology for composite component discovery. We explain in section 4 how mapping is done between the discovery ontology and the shared ontology. In conclusion, we will suggest some openings and prospects related to this study.

2 THE DISCOVERY AND THE INTEGRATION ONTOLOGY

We describe the semantics of components to express knowledge about functional and non-functional aspects of a component. This knowledge comprises:

- The structural aspects that specify the component's internal structure. The developer uses these aspects to determine if interaction exists between component operations and other components used to build the current project.
- The functional aspects that identify the functionalities of the component is expected to provide through many features. These features include methods that are used to adapt the behavior of the component to his context. The adaptation is made by specializing and customizing. The other kinds of features are used by the application specific part of a component-based software. Generally this type of information is specified by the component's methods.
- The non functional aspect specifies the component constraints related to communication or computation. The non functional aspect includes features such as performance, availability, reliability, security, adaptability and dependability. We distinguish static and dynamic categories of non functional features. Static features, such as security-related constraints, do not change during component execution. Dynamic features, such as performance-related properties, depend on the deployment environment.

All these features represent different and complementary views of a component. The feature set used to describe a component, depends on the developer action: discovery and integration. The discovery of a component is made by sending a query to the repository manager. Once a set of components has been selected, additional features are specified to select a component before integration. For the discovery action, the query includes functional and/or non functional features. For integration action, the structural features have to be specified.

The underlying approach for SEC++ is based on the following ontologies(see Figure 1): (KDJ10)

- The discovery ontology that specifies functional and non functional features.
- The integration ontology that describes the problem solving method (PSMs) used to specify the component's structural features.

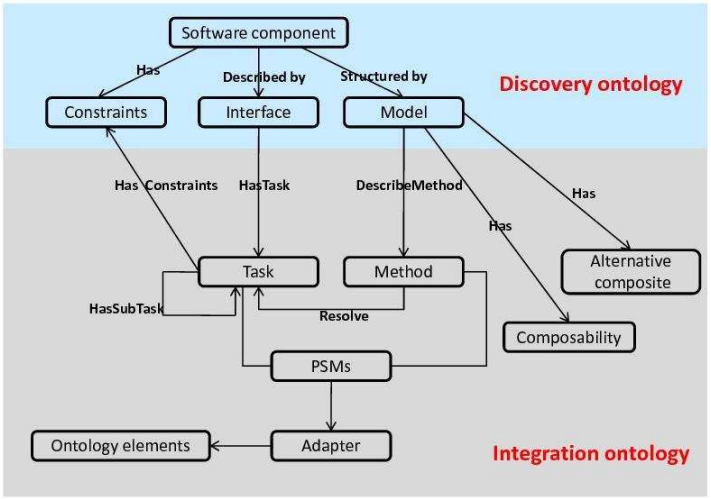

Figure 1: Discovery and integration ontologies

3 THE SHARED ONTOLOGY FOR COMPOSITE COMPONENT DISCOVERY

This approach exploits the advantages of semantic composition approaches, powered by ontologies at both component discovery and integration levels. Building on top of that, we introduce an ontology-based semantic approach. First, the semantic component specification provides a mechanism to enrich atomic components with more semantics than the syntactical method. Second, mapping atomic components and other relevant concepts into a centralized shared ontology offers a knowledge repository for software components (see Figure 2). The objective of semantic enhancement is to support ontological heuristics in order to enable automated and dynamic component composition. When our enhanced search engine SEC++ receives a query from a consumer, it first searches the discovery ontology. Our approach enhances the discovery ontology with a shared ontology. This centralized ontology represents relevant components

and concepts in a specific domain, constructed by mapping and integrating individual discovery ontologies for software components. Here, the ontological heuristics serves as guidelines to respond to a developer request. After using ontological heuristics on the shared ontology, SEC++ generates a number of alternative solutions to component composition. These alternatives are then evaluated by a decision engine using a set of criteria specified by the developer. Such criteria may include QoS-based optimization of component composition, business rules and strategies. A selected optimal composition scheme is then executed.

As for the integration ontology, we employ problem solving method to develop a local ontology for component. In the integration ontology we try to divide the component process into tasks. Tasks are either solved directly (by means of primitive methods), or are decomposed into subtasks (by means of decomposition methods). We use the Unified Problem-Solving Method Language (UPML) to describe the components of PSMs (task, method and adapter). Similarly, the component model subclass is especially beneficial for composition (see Figure 1). The proposed approach utilizes the component model class in two ways. For base components, a component model keeps information about composability, which specifies when the component can be used in a composite component. For composite components, a component model maintains alternative composite solutions incrementally for reuse. This semantic enrichment provides a self-learning capability of component composition.

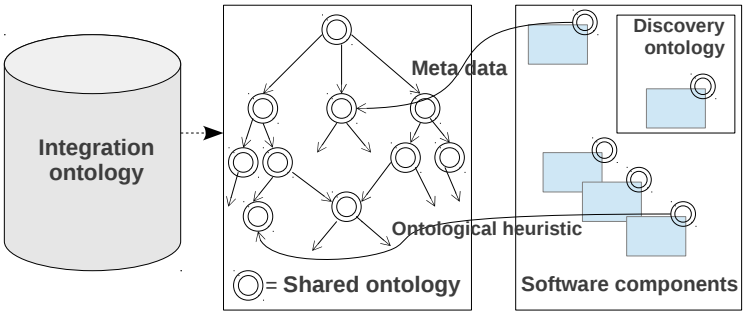

Figure 2: An ontology-supported system for component composition

Local integration ontologies are consolidated in a server by ontology mapping and integration. As a result, all relevant concepts and components in a domain are in the shared ontology, local discovery ontology for a component is mapped into the shared ontology, appearing as a node in the ontology tree. How to organize all components into the repository depends on domains and application requirements. For example, for calculating Matrix we can maintain semantic relationships (e.g., hierarchical and sibling relationships) between Matrix operations. The shared ontology also represents other application-specific concepts for mapping and integrating components. The mapping and integration not only unite component descriptions and concepts but also add more semantics. Moreover, the shared ontology enables ontological heuristics, thus facilitating dynamic component composition. For example, we can study composability of components based on some generic concepts. As a simple example, when composing component C2 that calculates the determinant of a real matrix by receiving the output parameters of a component C1 that calculates the sum of two matrix which have a natural type. At first glance, these two components cannot be composed. However,

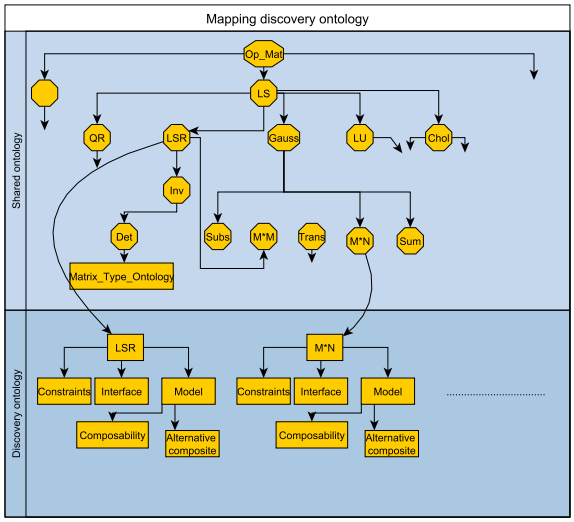

Figure 3: mapping the discovery ontology individual instances into shared ontology

the relationship between real and natural is revealed in the type ontology: natural is included in real.

4 MAPPING THE DISCOVERY ONTOLOGY INTO INTEGRATION ONTOLOGY

In this section, we present a prototype system for mapping a discovery ontology of mathematical service into integration ontology, which is developed based on the proposed ontology-supported software component integration. Our prototype system employs W3C-recommended standards (i.e., RDF+OWL) for semantic description and ontological engineering. The software utilized for this task is Protégé 3.4.4. At the discovery ontology description level, the prototype system translates component descriptions and then adds more semantics in the component model class. The composability property of the component model class can have values denoting possible ways for component composition.

Taking the Linear system resolution component as an example, its composability contains a list of possible parameter flows (from inputs to outputs), each of which can be a part of an alternative path in a composite component (Complex Matrix $\longrightarrow$ Complex Matrix), (Real Matrix $\longrightarrow$ Real Matrix).

Another way to exploit composability is first to attach composability to other properties with concrete meanings, then associate composability with composition rules.

all individual discovery ontologies are mapped together, appearing as nodes or subclasses in the shared ontology. For example, the LSR component is a subclass of the LS class (see Figure 3). The prototype can extract some metadata from individual discovery ontologies and map into the shared one. After organizing those base Matrix Operation services into a shared knowledge repository, the prototype adds other concepts relevant to Matrix operations, either domain-specific or generic, such as Type.

5 CONCLUSION

In this paper, we presented the latest version of SEC: SEC++, which cover the whole set of ontologies. It is a persistent component for discovering atomic and composite components. It delivers services and helps the developer to locate appropriate components for integration into the current work. To ensure the successfulness of this process, we developed the discovery, the integration and the shared ontologies. The ontological heuristics in used to enable automated and dynamic component composition. To improve the integration process we have used Output-Matching-Service and Input-Output-convertor service types used in matching parameters. In the future work, we plan to develop a fuzzy search system that aims to discover component which is most relevant to the user according to his/her perception.

REFERENCES

Rama Akkiraju, Carine Bournez, J.B. Domingue, Joel Farrell, Laura Ferrari, and Laurent Henocque. *Semantic Annotations for* WSDL *and* XML *Schema*, W3C recommendation edition, 2007.

Tarak Chaari and Kaouthar Fakhfakh. Semantic modeling and reasoning at runtime for autonomous systems engineering. In *9th International Conference on Ubiquitous Intelligence and Computing & Autonomic Trusted Computing(UIC/ATC)*, 2012.

Tarak Chaari, Said Tazi, Mohamed Jmaiel, and Khalil Drira. ODACE SLA: Ontology driven approach for automatic establishment of service level agreements. *IJSSOE*, 1(3), 2013.

Jos de Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Uwe Keller, and Michael Kife. *Web Service Modeling Ontology (*WSMO*)*, W3C recommendation edition, 2005.

Martin Junghans and Sudhir Agarwal. Web service discovery based on unified view on functional and non-functional properties. In *Proceedings of the 4th IEEE International Conference on Semantic Computing (ICSC 2010)*. IEEE Computer Society, 2010.

Sofien Khemakhem, Khalil Drira, and Mohamed Jmaiel. An integration ontology for components composition. *In International Journal of Web Portals*, 2(3):35–42, 2010.

Sofien Khemakhem, Khalil Drira, and Mohamed Jmaiel. *Modern Software Engineering Concepts and Practices: Advanced Approaches*, chapter Description, classification and discovery approaches for software components: a comparative study. IGI publisher, 2011.

Seog-Chan, Dongwon Lee, and Soundar R. T. Kumara. A comparative illustration of AI planning-based web services composition. *SIGecom Exch.*, 5(5):1–10, January 2006.

Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL *Web Ontology Language Guide*, W3C recommendation edition, 2004.