A Lightweight Formal Approach for Component Reuse

Khai T. Huynh*, Thang H. Bui, and Tho T. Quan

Abstract. Component reuse is playing a crucial role in today software design. However, the current approaches used in industry are quite effort-consuming due to the lack of an effective mechanism to describe and capture semantics in software components. In this paper, we propose a formal approach to overcome this problem, which is based on First-Order Logic (FOL). In one hand, FOL is sufficiently expressive to describe semantics in various software domains, from generic to specific ones. In the other hand, FOL also supports automatic searching, matching and inferring mechanism by computer-based tools and provers. Thus, our approach both effectively supports expert human to describe system components and computer programs to reuse those components, and even to compose new components from existing ones for further usage. We realize our approach as a framework which can be applied in various situations of software designs, as illustrated in some case studies.

Keywords: Component-based development, rapid software development, formal specification, component reuse.

1 Introduction

Component reuse is one of the most promising field of software engineering [1]. Enhanced productivity (as less code needs to be written), increased quality (since assets proven in one project can be carried through to the next) and improved business performance (lower costs, shorter time-to-market) are the main benefits of developing software from a stock of reusable components [2][3].

Khai T. Huynh · Thang H. Bui · Tho T. Quan
Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology
Ho Chi Minh City, VietNam
e-mail: 551220035@stu.hcmut.edu.vn

* Corresponding author.

V.-H. Nguyen et al. (eds.), *Knowledge and Systems Engineering,*
Advances in Intelligent Systems and Computing 326, DOI: 10.1007/978-3-319-11680-8_41

However, there are problems associated with reuse. There is a significant cost associated with understanding whether or not a component is suitable for reuse in a particular situation, and in testing that component to ensure its dependability. These additional costs will increase the overall development costs. There are some problems associates with reuse software components [3] as follows: (i) current software tools do not support development with reuse; (ii) the cost for creating, maintaining, and using a component repository is typically quite expensive; and (iii) there is currently lacking a practical and formal mechanism for finding, understanding, and adapting reusable components. The common reason for all of above problems is the lack of an effective mechanism to encode semantics to the descriptions of software components. There are several attempts reported to formally describe software design with embedded semantic information [4][5]. However, those approaches typically suffered from high computational cost thus causing some difficulties when applied in real life applications.

This observation urges us to consider an approach which should be at the same time conveniently lightweight for the human users and formally expressive for machine understanding and processing. This idea is realized as a framework proposed in this paper, which formally facilitates reusing existing components previously developed in a lightweight manner. It is achieved by the following technical aspect:

- Using first-order logic-based (FOL) [6] to describe components.
- Using machine learning-based approach for similarity evaluation between FOL formulas.
- Employing and enhancing the Artificial Intelligence (AI) planning technique to compose a design fulfilling an input request. During doing so, additional components can be composed and added to the current component library.

The rest of the paper is organized as follows. Section 2 discusses some background knowledge. Section 3 presents our proposed framework for rapid software development based on reusing formal specification components. Section 4 illustrates the capability of framework through two case studies. We present the related works in Section 5 and conclude the paper in Section 6.

2 Background

2.1 FOL-Based Component Specification

First-order logic (FOL) [7] is a well-known formal representation which extends the classical proposition logic with predicate, quantifier and variable. Thus, FOL allows more flexible and compact representation of knowledge [8]. In our framework, FOL is used to specify software components. The component is considered as a black box and stored in repository as a structure in Listing 1 with *expression* is a FOL-based expression.

Listing 1: Structure of a component stored in repository.

```
<component name=comp_name>
   <input>variable: type_name</input>
   <output>variable: type_name</output>
   <pre>expression</pre>
   <post>expression</post>
</component>
```

Listing 2 shows an example specification of *binarySearch* function considered as a component stored in the repository. The components are organized in a hierarchical organization to provide a faster means for browsing and searching.

Listing 2: Specification of binarySearch function.

```
<component name=java.util.Arrays.binarySearch>
  <input>a: Object[], x: Object</input>
  <output>i: Integer</output>
  <pre><![CDATA[
     a != null && (\forall i:int; 0<i && i<a.length; a[i-1]<=a[i])
    ]]></pre>
  <post><![CDATA[
    ((\forall j:int; 0<=j && j<a.length; a[j]!=x) && i==-1) ||
     (\exists j:int; 0<=j && j<a.length; a[j]==x) && i==j))
   ]]></post>
</component>
```

2.2 FOL-Based Similarity Computation

Once software components are represented as FOL formulas stored in a repository, it is needed to develop a similarity measure between those formulas to support searching the required component over the repository. To achieve this, we rely on the following definitions [9]:

Let Γ be the set of FOL formulas, $T = \{t_1, ..., t_m\}$ is set of all symbols and terms that appear in Γ and c, p are two FOL formulas in Γ.

Definition 1. Feature matrix

The feature matrix $\Phi = \Gamma$ x $\{1, ..., m\} \rightarrow \{0, 1\}$

$$\Phi(c, i) = \begin{cases} 1 \text{ if } t_i \text{ appear in } c \\ 0 \text{ otherwise} \end{cases}$$

Definition 2. Feature function

The feature matrix gives rise to the *feature function* φ.

Define φ: $\Gamma \rightarrow \{0, 1\}^m$ which for $c \in \Gamma$ is the vector φ^c with entries in $\{0, 1\}$ satisfying:

$$\varphi_i^c = 1 \leftrightarrow \Phi(c, i) = 1$$

Definition 3. Classifier function

For each p in Γ, the classifier function is defined:

$$C_p(.) : \Gamma \rightarrow R$$

Given a conjecture c, $C_p(c)$ estimates how useful p is for proving c. The classifier function is useful when we need to decide if a premise p would then be useful to prove a required goal c. It would be the case if $C_p(c)$ is above certain threshold. A common approach to ranking is to use classification, and to combine the real-valued classifiers [10]. The premises for a conjecture c are ranked by the values of $C_p(c)$, and we choose a certain number of the best ones.

With vector φ has been calculated, we can develop methods of calculating the similarity $C_p(c)$. These methods can be as simple as the angle between the vector calculus, or others methods such as Naive Bayes [11], depending on the complexity of the problem.

Listing 3: Specification of three components: f1, f2 and f3.

```
<component name=f1>
  <input>x: double</input>
  <output>y: double</output>
  <post>y== 2 * x</post>
</component>
<component name=f2>
  <input>x: double</input>
  <output>y: double</output>
  <post>y==-x</post>
</component>
<component name=f3>
  <input>x: double</input>
  <output>y: double</output>
  <post><![CDATA[(x>0)=>(y==x+1) && (x<=0)=>(y==log(x))]]></post>
</component>
```

For example, with the components $f1$, $f2$ and $f3$ are described in Listing 3, if we denote the terms p_1, p_2, p_3 and p_4 corresponding to the set of operators: {+, -}, {*}, {>, <=} and {log}, we will have the feature matrix as follow:

φ	p_1	p_2	p_3	p_4
$f1$	0	1	0	0
$f2$	1	0	0	0
$f3$	1	0	1	1

Then, we will calculate the feature vectors for each component based on the matrix φ. For example, we have the vector representing the component $f2$ is φ^{f2} = [1,0,0,0] (value 1 in ith column mean that $f2$ has the feature p_i). With the calculated vectors φ^{fi}, we can develop the techniques for calculating the similarity between two vectors. This technique can be as simple as calculation the angle between two vectors, or maybe a machine learning method such as Naive Bayes [11], depending on the complexity of each domain.

2.3 FOL-Based AI Planning for Component Composition

AI planning is the area of study concerned with the automatic generation of a plan to solve a problem within a particular domain. At its simplest, a plan is a sequence of actions. Given an initial state, the planner tries to find the actions required to achieve some goal conditions [12].

According to the STRIPS representation [13], the system consists of a set of *states*, *goals* and *operations*. Each operation has *operation name*, *preconditions* - a sentence describing the conditions that must occur so that the operator can be executed and *effect* - a sentence describing how the world has change as a result of executing the operator. The algorithm of classical AI planning with STRIPS representation is presented in [13].

With classical AI planning, the matching mechanism between the two FOL formulas is exact matches. In fact, the matching is not so simple. It is easy to see that in the mathematical formulas representation, two different formulas may have the same computation result but there are many different representation ways, example ($2*x$) and ($x+x$). Hence, to make the matching is more precise, we enhance the classical AI planning that allows to prove the truth of two FOL formulas have different representation with a theorem prover, such as Z3 [14].

3 The Framework

Fig.1 describes the framework for automatic search and reusage components based on formal specification. The framework has three modules as follows.

3.1 Component Selection Module

This module acts as a pre-processing step of framework. It computes the similarity between the logic formulas representing the components, as discussed in Section 2.2. Thus, we can narrow the component search space. The narrowing is obtained by similarity computation between formal specifications of the components in the repository.

3.2 Component Composition Module

The second module of framework is the component composition module. This is the most important part of framework. The searching and synthetic components module based on AI-planning works on the following principle. Starting from initial requirement (c) and the set of components (P) which have a degree of similarity in a predefined threshold for (c), the planner will choose a component p from P (with the priority of the degree of similarity). As discussed in Section 2.3, we rely on a mathematical prover to judge if the precondition of p is equivalent to that of c. If it is the case, the search process stops and returns results (return p). In contrast, the

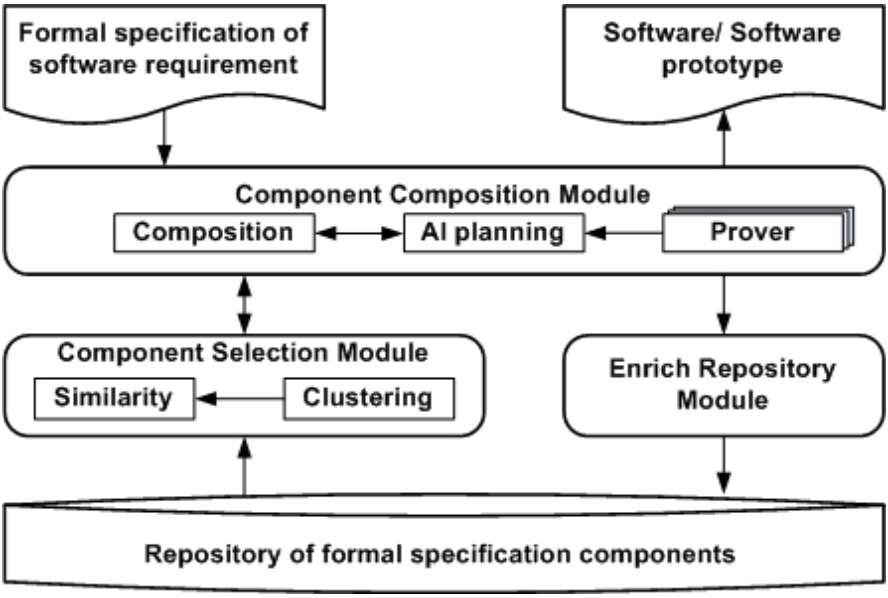

Fig. 1 Structure of framework

algorithm will consider the precondition of p as new postcondition and the search process will repeated. In case of deadlock, the algorithm will backtrack with the function p' in P ($p' \neq p$).

3.3 Enrich Repository Module

This module of framework performs enriching the component specification repository for future use. So, this module helps us to optimize the system performance when encountering a search request that is similar to those previously performed.

4 Case Studies

4.1 Apply the Framework to Find Function(s) Satisfy the Requirement

In this section, we present an illustrative example of the automatic API functions searching which satisfy the requirement specifications.

We want to generate a program segment which returns the list of students who to be received scholarship from a list of students. List of students received scholarship is defined as the list which has a maximum of n items (usually, n is 10% of total students) taken in order from top to bottom by GPA; The student in this list must have a GPA greater than or equal 7.0/10.0 and every subject must has the result is

greater than or equal 5.0/10.0. With that requirement, the specification is given as follows:

```
//@in s: StudentList, n: int
//@out x: StudentList
//@require listnotnull(s)
//@ensure x.len<=n && orderlist(x) &&
//@  (forall i: int; i>=0 && i<x.len; x[i].GPA>=7.0) &&
//@  (forall i: int; i>=0 && i<x.len;
//@    (forall j:int; j>=0&&j<x[i].subjs.len; x[i].subjs[j]>=5.0))
```

In our system repository, we have a list of API functions:

The function returns the list of students who have the GPA greater than or equal the value *n*:

```
<component name=get_StudentList_By_GPA>
  <input>s:StudentList, n:float</input>
  <output>x: StudentList</output>
  <pre>listnotnul(s)</pre>
  <post><![CDATA[ (forall i: int; i>=0 && i<x.len; x[i].GPA>=n)
      ]]></post>
</component>
```

The function returns the list of students who have the result of every subject is greater than or equal the value *n*:

```
<component name=get_StudentList_By_SubjectScore>
  <input>s:StudentList, n:float</input>
  <output>x: StudentList</output>
  <pre>listnotnul(s)</pre>
  <post><![CDATA[ (forall i: int; i>=0 && i<x.len;
        (forall j:int; j>=0&&j<x[i].subjs.len; x[i].subjs[j]>=n))
      ]]></post>
</component>
```

The function returns the first *n* students from the list of Student. If the number of items in student list is less than *n*, all item of student list will be returned.

```
<component name=get_n_Elements>
  <input>s:StudentList</input>
  <output>x: StudentList</output>
  <pre>listnotnul(s)</pre>
  <post><![CDATA[ x.len<=n ]]></post>
</component>
```

The function sorts the list in descending order of GPA:

```
<component name=sort_DescList_By_GPA>
  <input>s:StudentList</input>
  <output>x: StudentList</output>
  <pre>listnotnul(s)</pre>
  <post>orderlist(x)</post>
</component>
```

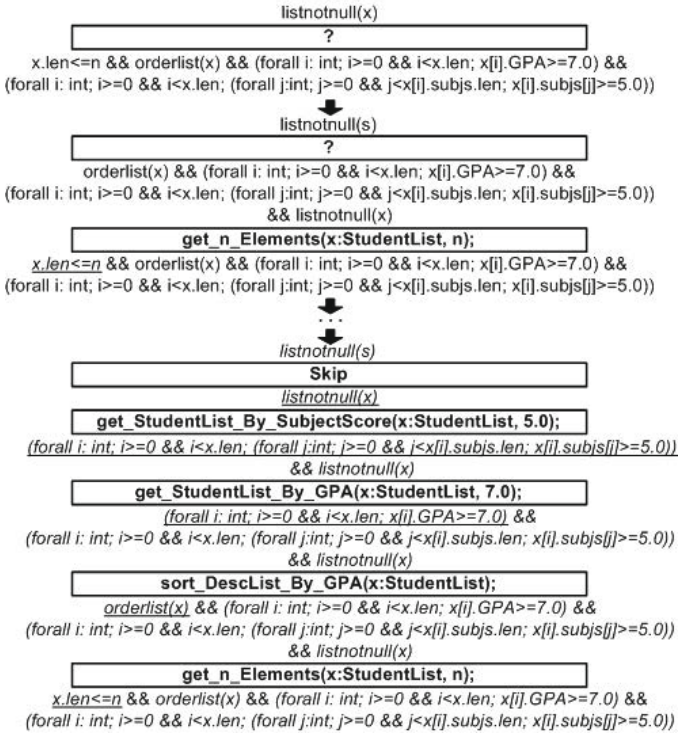

Fig. 2 Planning process to generate list of Student received scholarship

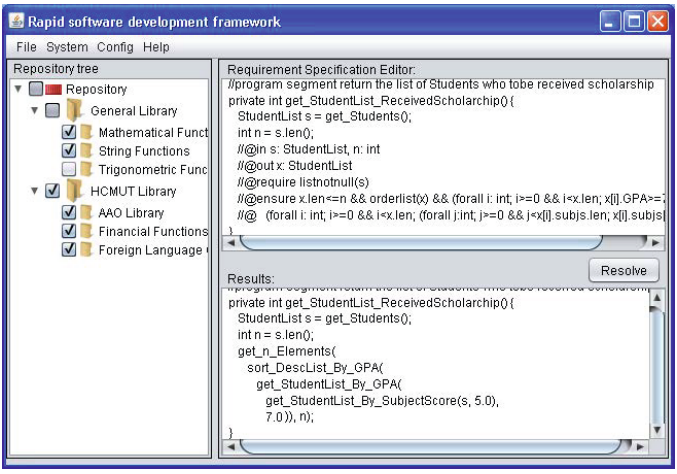

Fig. 3 Illustrated result of generating list of Student received scholarship

With the above requirement and API function repository, the planning steps perform as in Fig.2 and Fig.3 illustrates the result produced from framework. Note that the composed result can be further added to the repository for future reuse if needed.

4.2 Automatically Searching for Web Template

In practice, when developing a website, we want to find the templates which can apply to our website. Searching the template matching with the requirement is a boring and time consuming work. With our framework, we can perform this searching process automatically. Suppose that we have a repository of website templates which contains three specified templates as follows:

Template_1 has three parts: *header*, *body*, and *footer*. The *header* contains *banner*, *logo* and *menu*. *Body* of the template is divided into three columns: *left_col*, *center_col* and *right_col*. The *menu* contains three links: *Home*, *Intro* and *Contact*. This template is specified:

```
<component name=Template_1>
  <post><![CDATA[ page(header, body, footer) &&
     header(banner, logo, menu) &&
     body(left_col, center_col, right_col) &&
     menu(Home, Intro, Contact) ]]></post>
</component>
```

Template_2 has three parts: *header*, *body*, and *footer*. The *header* contains *banner* and *logo*. *Body* of the template is divided into three columns: *left_col*, *center_col* and *right_col*. The *left_col* contains a *vertical_menu* which has three links: *Home*, *Intro* and *Contact*.

```
<component name=Template_2>
  <post><![CDATA[ page(header, body, footer) &&
      header(banner, logo) &&
      body(left_col, center_col, right_col) &&
      left_col(vertical_menu)) &&
      vertical_menu(Home,Intro,Contact) ]]></post>
</component>
```

Template_3 has three parts: *header*, *body*, and *footer*. The *header* contains *banner* and *logo*. *Body* of the template is divided into two columns: *left_col* and *main_col*. The *left_col* contains a *vertical_menu* and the *vertical_menu* has three links: *Home*, *Intro* and *Contact*. This template is specified:

```
<component name=Template_3>
  <post><![CDATA[ page(header, body, footer) &&
      header(banner, logo) &&
      body(left_col, main_col) &&
      left_col(vertical_menu)) &&
      vertical_menu(Home,Intro,Contact) ]]></post>
</component>
```

User wants to find a website template which has three parts: *header*, *body* and *footer*. The *header* contains the *banner* and *logo*. Body of the site is divided into three columns: *left_col*, *center_col* and *right_col*. The *left_col* contains a vertical menu which has links: *Home*, *News*, *Products* and *Contact*. The specification of user requirements is as follows:

```
//@ ensure page(header, body, footer) && header(banner, logo) &&
//@    body(left_col, center_col, right_col) &&
//@    left_col(vertical_menu)) &&
//@    vertical_menu(Home,News,Products,Contact)
```

With above requirement, the similarity computation between the specification of requirement and templates in repository will return the *Template_2* is the most suitable template with a similarity degree of 80% (matching 4 of 5 terms). The *Template_1* has a similarity degree of 40% (matching 2 of 5 terms) and the *Template_3* has similarity degree of 60% (matching 3 of 5 terms).

Thus, the *Template_2* is the needed template. Of course, this template does not completely satisfy to the requirement, but only the one that the best fits the requirement. Then, user may need to further modify before deploying.

5 Related Work

Intelligent Software Architecture Reuse Environment (ISARE) [15] is a framework on the aspects of software Architecture on quality attributes. It is mainly directed on reuse of available software architecture evaluation methods. ISARE strengthen the Software Architecture repository and automate the software architecture selection and evaluation methods for efficient and reliable systems. The main input of ISARE is the set of architectural styles and their comprehensive analysis against the set of quality attribute.

Trusted National Software ResoUrce Sharing and Co-operaTeng EnvironmEnt (TRUSTIE) [16] is a tool that is used to construct a large-scale software production environment for trust worthy resource sharing and development cooperation. TRUSTIE is used as a command-line tool for component development. A new tool has to be developed to provide advantages of meta-model to model-editor generators. A research issue presents the close connection with Service Oriented Architecture [16].

N Md Jubair Bash et al. [17] has proposed a framework studio for effective component reusability which provides the selection of components from framework studio and generates source code based on the stakeholders needs. This framework studio is implemented in using swings which are integrated onto the Netbeans IDE which helps in faster generation of the source code.

6 Conclusion

In this paper, we propose a framework which allows automatic reuse and composition of software components. In our framework, we first suggest to use First-order Logic (FOL) for describing software components. By doing so, our frameworks can serve various situations of software design, ranging from function APIs to GUI templates searching. One of the key properties of our framework is that it can be applied in a lightweight manner, i.e. users are not restricted to a certain convention when using FOL to describe the components. Then, due to the solid mathematical foundation of FOL, the predefined FOL-based components can be processed effectively for searching, matching or even incremental composition by computer-based tools and provers. We have demonstrated the effectiveness of our framework in two case studies reflecting two common demands of component reuse in practice, which are Java API and Web interface searching.

References

[1] Basili, V., Rombach, H.D.: Support for comprehensive reuse. IEEE Software Engineering Journal, vol 6(5), 303–316 (1991)
[2] Sametinger, J.: Software engineering with reusable components. Springer, New York (1997)
[3] Sommerville, I.: Software Engineering, 9th edn., ch. 16. Addison-Wesley (2010) ISBN-10: 0137035152
[4] Liu, S., et al.: SOFL: A formal engineering methodology for industrial applications. IEEE Transactions on Software Engineering 24(1), 24–45 (1998)
[5] Zschaler, S.: Formal specification of non-functional properties of component-based software systems. Software & Systems Modeling 9(2), 161–201 (2010)
[6] Barwise, J.: An introduction to first-order logic. Studies in Logic and the Foundations of Mathematics 90, 5–46 (1977)
[7] Srivastava, S.M.: Syntax of First-Order Logic. A Course on Mathematical Logic, pp. 1–13. Springer, New York (2013)
[8] Ayorinde, I.T., Akinkunmi, B.O.: Application of First-Order Logic in Knowledge Based Systems. African Journal of Computing & ICT 6(3) (2013)
[9] Jesse, A., et al.: Premise selection for mathematics by corpus analysis and kernel methods. Journal of Automated Reasoning, 1–23 (2011)
[10] Richard, M.D., Richard, P.L.: Neural network classifiers estimate Bayesian a posteriori probabilities. Neural Computation 3(4), 461–483 (1991)
[11] Irina, R.: An empirical study of the naive Bayes classifier. In: IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, vol. 3, pp. 41–46 (2001)
[12] Russell, S.: Artificial intelligence: A modern approach, 2nd edn., vol. ch. 11. Pearson Education India (2003)
[13] Richard, E., Nilsson, J.: Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 189–208 (1971)
[14] de Moura, L., Bjørner, N.S.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
[15] Rizwan, A., Saif, R.K., Aamer, N., Tai-hoo, K.: IASRE: An Integrated Software Architecture Reuse and Evaluation Framework. In: ASEA 2010, pp. 174–187 (2010)

[16] Xinyu, Z., Li, Z., Cheng, S.: The Research of the Component-based Software Engineering. In: Sixth International Conference on Information Technology: New Generations, pp. 1590–1591. IEEE (2009)
[17] Jubair, B.N.M., Salman, A.M.: A Framework Studio for Component Reusability. CS & IT-CSCP-2012, 325–335 (2012)