TECHNOMICS: APPROACH TOWARDS BEST QUALIFIED COMPONENT

Vishnu Sharma
Faculty: School of Computer & Systems Sciences
Jaipur National University
Jaipur, India

Vijay Singh Rathore
Director:
Shri Karni College
Jaipur, India

Abstract— *The focus of this paper is to suggest a method which enhance the adaptability of available components. This paper suggests an architecture which involves traditional (Keywords based) and advanced techniques to search a component. If still suitable component is not found then and few changes in the architecture of component are required, a user may suggest the changes. This response is sent to server. Server lets this response to be implemented based on some properties of the available component. Improved component is verified with available models. If it passes this verification step. A new qualified component is made available to user to download. It can be downloaded and implemented in user projects.*

Keywords-component; formatting; style; styling; insert (key words), MVC(Model View Controller), SQL(Structured Query Language), Technomics Compiler, IA(Interface Automata),reuse, Component Retrieval Technique (CRT), Product Line Architecture(PLA), CBSE(Component Based Software Engineering),SDLC(System Development Life Cycle)

I. INTRODUCTION

This approach is useful for the software developers to obtain appropriate components to develop efficient software without wasting lot time in developing from stretch and test. It also provides flexibility to user to obtain the best suitable component without going through large development process of component at his own. An efficient way to retrieve appropriate component from repository. The suggested architecture effectively supports query specification and component search. It further guides users to exploit component resources for reuse.

In today's scenario most of the Software Development firms businesses are doing component based production. On the same pattern to augment the productivity, quality and efficiency of the software and reduce efforts done on testing Component Based Software Engineering has taken its market place. Components are piece of code which are ready to be embedded in our software without worrying about the failure. Developing any complex software from scratch is expensive. Development and testing process of all

component is time consuming. If the development task is not performed in a proper manner without testing in early stages of SDLC, it may easily get out of control making it almost impossible to debug and even more difficult to modify the code when the code goes 100 KLOC or more. Still problem to get the best qualified component is disquieting the developers. In this paper a proposed architecture works to provide the developers flexibility to convert the available components in more qualified components.

II. SOFTWARE COMPONENT REPOSITORY

Software components enhances the capability of software The software architecture is one of the main object developed during the software life cycle [1] because it determines most of the non-functional characteristics the resulting software will have, and it is also one of the most difficult documents to change once the software is deployed [2]. Component-Based software engineering is the key technology to cope with the requirements of high productivity, low maintenance cost and reliability of software products [6]. Software product lines are a trend for planned colossal reuse of software assets [3]. The most typical reusable assets are software components, but we can also reuse the product line architecture (PLA), software requirement documentation, and test cases, among others. The PLA is an important reusable asset because all software products in the family share the same design [4]. Therefore, the PLA design should be carefully approached making sure it will produce software that complies with the desired requirements.

There are major steps identified to obtain a best qualified component:

a) Search of relevant component from repository.
b) Testing of component for relevancy.
c) Suggest changes in the certain properties of the software.
d) Implementation of suggested changes.

e) Verification against certain available models.
f) Communicating the qualified component to developer for downloading purpose.

Finding the component includes a major area of search techniques and retrieval techniques. In this research paper we will try to provide a framework to get best qualified component involving flexibility to modify component up to certain level..

III. COMPONENT RETRIVAL TECHNIQUES(CRT)

a) Few traditional approaches to get best qualified component are :
b) Keyword search requires assigning to each object a number of relevant keywords or indices [5].
c) Full-text Retrieval : when a person wants information from that stored collection, the computer is instructed to search for all documents containing certain specified words and word combinations, which the user has specified [5].
d) Hypertext Search: The basic building blocks in hypertext are nodes and links. Each node is associated with a unit of information, and nodes can be of different types[5].
e) Enumerated classification: Enumerated classification uses a set of mutually exclusive classes, which are all within a hierarchy of a single dimension [6].
f) Attribute value: The attribute value classification scheme uses a set of attributes to classify a component [6].
g) Faceted: Faceted classification schemes are attracting the most attention within the software reuse community [6, 7].
h) Signature matching : Consider the signatures presented in Figures 1 and 2 for a stack of integers and a queue of integers, respectively [8]

IV. TECHNOMICS : ARCHITECTURE

There are four modules that contains with our prototype system implementation, see Figure 1, these are:

i. User Interface
ii. Database
iii. Technomics, and
iv. Check Authenticity

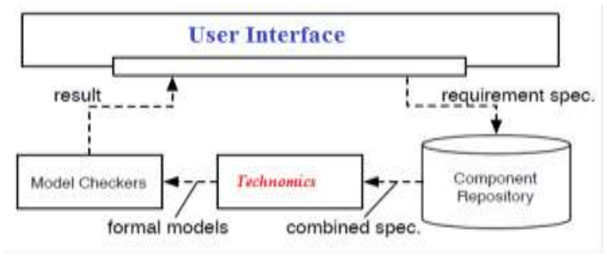

Figure 1: System Modules

Web interface is a medium through which the user interacts with the system. A representative use of Figure 1: The System Modules are as follows:

1. The user gets a list of components by searching it using keyword. The list of candidate components is displayed according to typed keyword.
2. The specifications of the candidate components are displayed and the requirement specifications can be modified accordingly.
3. The requirement specification of component repository is uploaded by the user. That required specification is combined with each candidate component specification and the TECHNOMICS translator is called to translate the combined TECHNOMICS specification to the models in some existing formal language.
4. The relevant model checking tools are called by the system to check whether the component requires behaviour.

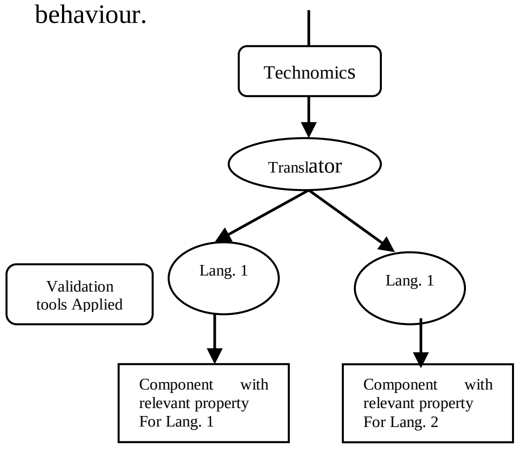
Technomics
Translator
Validation tools Applied
Lang. 1
Lang. 1
Component with relevant property For Lang. 1
Component with relevant property For Lang. 2

Figure 2: Implementation of Technomics

In our prototype system, the TECHNOMICS translator is executed in .Net MVC. The web interface is implemented by .Net Framework which is running on SQL and is used to build the sample component repository.

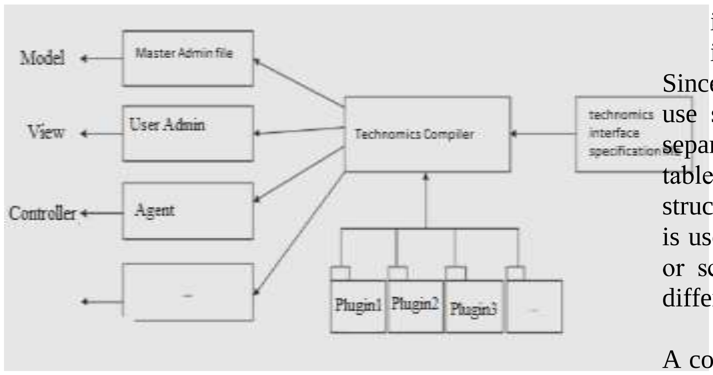

Figure 3: TECHNOMICS Translator

V. TECHNOMICS TRANSLATOR

The functioning of TECHNOMICS Translator is in such a way that the compiler of TECHNOMICS is fed with TECHNOMICS specifications. Using the help of various language plugins, complier translates the specifications of TECHNOMICS into existing formal languages, such as Master Admin (MA), C# [9], Interface Automata (IA) [10], etc. The translated formal models are entered to their accompanying checking tools. Following Figure 2 displays such kind of workflow. Following advantages contains with such design:

- To check the component compatibility, usually model checking tools can be used. A single TECHNOMICS specification is sufficient and can be reused with the variety of tools. If any new and more powerful tool becomes accessible, it is only needed to write a code generator to use within our framework.
- Component developers and users are free to exchange components and requirement description in TECHNOMICS without any tension about which checking tools that they are applying.

The prototype is assembled with plug-ins to support Master Admin and C# at the moment. These two tools have been selected to demonstrate the feasibility and applicability of our architecture.

VI. COMPILER

The compiler of TECHNOMICS is executed with the help of an open architecture. It authorizes several language modules to plug into the compiler. This enables the compiler to translate TECHNOMICS to other languages without recompilation of the compiler's source code. It is implemented in layers, see Figure 3.
In TECHNOMICS the combined specification comprises three parts:

i. Master Admin
ii. User Admin (Environment Components) and
iii. Agent (User Requirements)

Since different grammar rules apply for different parts, we use separate classes to grip grammar checking and make separate tables for each part. The tables contain symbol tables and rule tables. A "Symbol table" is a common data structure where every symbol present in the source code and is used by compiler is related with the information like type or scope level. A "Rule table" stores the rules related to different services that the component provides.

A common translation layer is present at the top of grammar checking. Its key task is to identify the different portions of the specification, and allot a relevant grammar-checking module to that part. Thereafter, a list of symbol and rule tables is generated which meets to the level of standard compilation process. A number of code generators is supported by TECHNOMICS framework. To load a plug-in class, the plug-in manager is called by the compiler and the plug-in class takes those intermediary tables as inputs and produces the translation for the plug-in language.

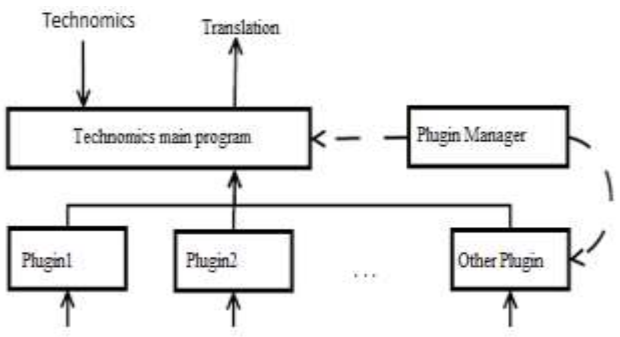

Figure 4: Layers of Compiler Implementation

VII. DEVELOPING PLUG-INS

- Making enable plug-ins has given users flexibility to deal with a diversity of modelling languages. While developing the TECHNOMICS compiler plug-ins, following steps was followed:
- Defines the mapping from the tables to the language which is supported by plug-in.
- Encompassing the common translation layer class to create the intermediate symbol tables and rule tables. All available plug-ins have the same tables.
- A plug-in property file is written and the Plugin Manager class locates it and load the plug-in at runtime.

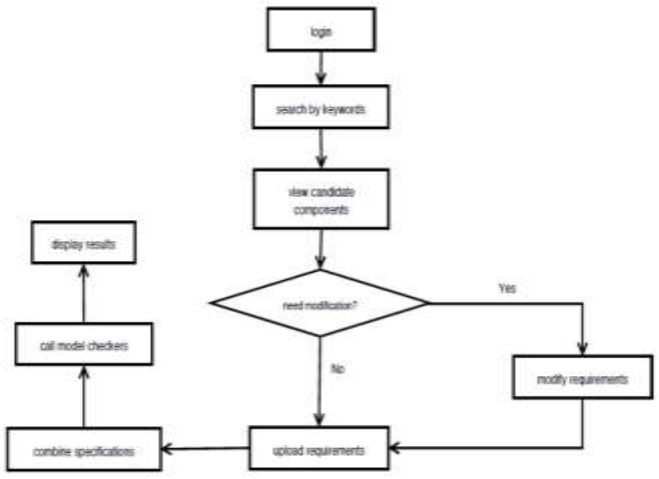

Fig 5: System Flowchart

Therefore it is imaginable to add support to various languages as long as the corresponding plug-ins is executed. However, before picking up a language, some concerns need to be considered, as: whether the transition system is explained; whether a matching theory is present to check compatibility of component. Once the selection of language is completed, then the major task is begun to define the mapping from TECHNOMICS to that language.

We have executed the plug-ins for C#. Both of them can be used to explain transition systems, although C# essentials to import the additional linear ordering module. For both tools, testing component compatibility is to test whether the composition of the two components have any prohibited behaviors. The component composition is directly supported by C# such as Component 1 or Component 2. A transition system in TECHNOMICS is decided by the guidelines of the services.

In the TECHNOMICS, input ports are read-only, but output ports can be altered. MA has a similar rule on the interface ports, therefore the translation from TECHNOMICS to MA is direct. C# does not have the idea of ports, supposing all the variables are writable. Therefore in C#, there is no necessity to declare variables as input or output.

The scenarios and properties can be described to check it in requirement specification. Scenarios could only be interpreted to MA which supports monitor automata and can implement along with the component. Properties are transferable to different assertions which is supported by both MA and C#.

VIII. COMPONENT REPOSITORY

There are two databases in the component repository:

- User Database: This database has all the information about the registered users of the repository, like as their usernames, passwords, contact details, etc. Again the users are classified as component users and component developers.
- Other Database: The name of component, its category, keywords used to describe the component and TECHNOMICS specification are stored in this database.
- Each component is also associated with a component developer.

WEB INTERFACE

Different user interfaces are applied for both component developers and users. Through this web interface, the system permits developers to add components, including specifying components in TECHNOMICS. The requirement can be uploaded by users and the system match components based on the requirement specification.

Fig. 5 depicts how the web interface can be used by the users and developers. A user searches components by inserting keywords and the components having some of the keywords from the inserted ones will be retrieved as candidate components for further behavioral specification matching.

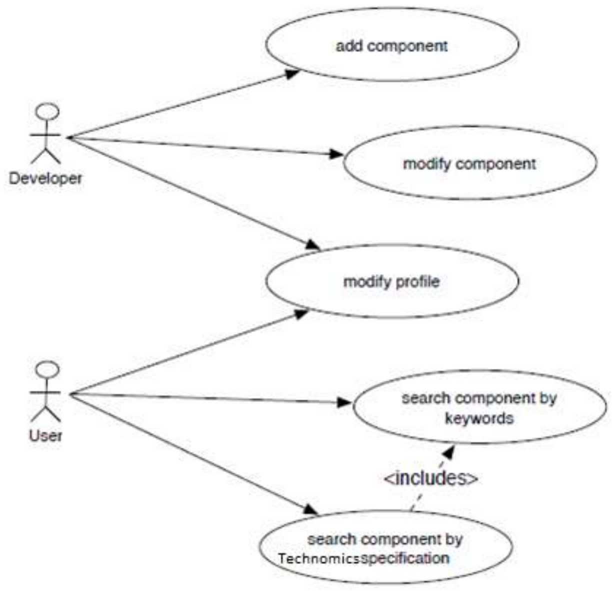
add component
Developer
modify component
modify profile
User
search component by keywords
<includes>
search component by Technomicsspecification

Figure 6: Use Case through Web Interface

The TECHNOMICS compiler compares candidate component specification and requirement specification. According to that parsing, it retrieves all the names, values, variables and services. If there is no parsing error, then the user is lead to where name mapping could be done.

Name mapping is not needed when the requirement specification is changed according to the component specification. However, the name mapping module is useful in checking the syntax of both specifications. Thereafter, the requirement specifications join with each candidate component specification.

The system chooses which language it should be interpreted to by testing if there are situations defined. If so, the collective specification will be interpreted into C#. However, if the properties are defined in the specification, it will be translated to both MA and C#. If it translates to C#, the scenario definition is ignored. The properties that are not sustained by both tools will be also ignored by the interpreter.

In the background, the system will run the model checkers. Therefore checking the model is transparent to the component user. In order to complete that, Java scripts is used for checking. For C#, the command interface of C# Analyzer is raised. On the basis of model checkers, the system displays whether the candidate components have required behavior.

FUTURE WORK

Our future work can be carried out in the following four directions:

- This chapter expresses the architecture and thorough implementation of our prototype component selecting system based on checking the behavioral compatibility. Presently, it only supports conversion to C# and some requirement specification work which can be done manually. There is some more future work where we can work upon like as adding more tools support, such as Ticc[11], and automatic tool selection becomes an issue.
- Since our approach has not achieved absolute automation due to the modeling unpredictability from people to person, somehow few manual work to accommodate user requirements is required when using our proposed prototype system. We will try to resolve this issue by defining a formal transformation from one model to another. On the Basis of this definition, all the models can be integrated.
- As the number of the model checking tools used in our proposed framework increase, it is mandatory to apply a trigger that will be able to automatically decide on the proper tools for checking component behavioral compatibility, as different available tools have their own features and support for checking different properties of component.

REFERENCES

[1] Martin Fowler. Who Needs an Architect? IEEE Software, 20(5):1H3, 2003.
[2] Len Bass, Paul Clements, and Rick Kazman. Software Architecture in Practice.SET Series in Software Engineering. Addison-Wesley, 2 edition, 2003.Immense
[3] Paul Clements and Linda M. Northrop. Software Product Lines: Practices and Patterns. Addison Wesley, first edition, August 2001.
[4] Jan Bosch. Design and Use of Software Architectures. Adopting and Evolving a Product Line Approach. Addison Wesley, first edition. May 2000.
[5] Tomas isakowitz and Robert j. Kauffman "Supporting search for reusable software objects. Center for Digital Economy Research Stern School of Business Working Paper IS-93-47
[6] Dr C.v.guru Rao and P.Niranjan "An Integrated Classification Scheme for Efficient Retrieval of Components". Journal of Computer Science 4 (10): 821-825, 2008 ISSN 1549-3636 © 2008 Science Publications
[7] P.Niranan and Dr.c.v.GuruRao. "A mock-up tool for software component reuses Repository"
[8] Rajender Nath, Harish Kumar; Building Software Reuse Library; 3rd International Conference on Advanced Computing and Communication Technology- ICACCT-08; Asia Pacific Institute of Information Technology, Panipat , India; November 08-09, 2008, pp. 585-587.
[9] Hsinchun Chen and Kevin J. Lynch, "Automatic Construction of Networks of Concepts Characterizing Document Database," IEEE T. Systems, Man, and Cybernetics 22 (5) (1992) 885-902.
[10] Haines, G., Carney, D. and Foreman, J, Component Development / COTS Integration, Software Technology http://www.sei.cmu.edu/str/descriptions/cbsd_body.html., 1[18] Harmon, P., Components, Component Deve Strategies, Vol. 8, No. 7, July 1998.
[11]Seacord, R., Hissam, S., Wallnau, C., Agora: A Search Engine for Software Components, CMU/SEI-98-TR-011, August 1998.