Rule-Based COTS Integration

Somjai Boonsiri

Faculty of Science, Chulalongkorn University
Bangkok, Thailand 10330
Somjai.B@chula.ac.th

Robert C. Seacord

Software Engineering Institute
Pittsburgh, Pennsylvania 15213
+1 412/268-7608
rcs@sei.cmu.edu

David A. Mundie

Software Engineering Institute
Pittsburgh, Pennsylvania 15213
+1 412/268-7394
dmundie@sei.cmu.edu

Component-based software development is considered to be a promising technology to increase software development productivity. However, developing component-based applications faces different challenges. One of them is identifying component ensembles that satisfy any particular system requirements specification. In this paper we introduce a component integration evaluation based on software engineering integration rules. These rules represent real-world experiences and are combined into knowledge base. These representative rules evaluate compatibility among components according to their attributes of component specification.

Keywords: rule-based COTS integration, ensemble evaluation, component-based software development, component composition.

The ACM Classification System: D.2.3 Reusable Software

1. INTRODUCTION

Component-based software development that assembles existing software components is considered to be a promising technology to increase software development productivity. Basili and Boehm (2001) have shown that more than 99 percent of all executing computer instructions come from COTS (commercial off-the-shelf) products. However, developing component-based applications faces several different challenges. One of them is identifying suitable component ensembles that satisfy system requirements specifications.

In evaluating and selecting components, it is becoming increasingly important to have a defined and potentially automated process for component evaluation due to the ever-increasing number of components, the broad range of factors effecting their integration, and the combinatoric ways in which these components can be combined into systems. Orso *et al* (1999) demonstrated the need

Manuscript received: 22 January 2002
Communicating Editor: Denis Warne

for metadata in the testing and analysis of distributed component systems. Similarly, metadata can be used to describe the contents of components for use in the evaluation process.

In this paper we describe an automated rule-based approach to evaluating ensembles of components within the context of a system requirements specification.

The structure of this paper is as follows. We introduce rule-based approach in Section 2 and describe the system overview in Section 3. In Section 4 we present the database structures used in this system and express the result from the experiment in Section 5. Potential future work and some concluding remarks are given in Section 6 and 7, respectively.

2. RULE-BASED APPROACH

A rule base or knowledge base is a collection of rules designed through dialogues between a domain expert and a knowledge engineer or domain experts themselves (Rarandi *et al*, 1986). To present this approach in component integration, we collect representative software engineering integration rules from domain experts in component integration and assert them into rule-based software called ILOG JRules (ILOG, 2000) – a commercial expert system designed to work with Java.

JRules is an object-oriented rule-based programming language. A JRules application consists of a set of rules and a collection of objects. Each rule is composed of a header, a condition part, and an action part. A rule is processed in the following manner: given a set of conditions, each condition with its variables is matched to an object with its field values. A condition may include tests on variable values. If all the conditions are matched (that is, the variables return field values that fulfill the tests), the action part may be executed. The set of actions may include simple operations such as printing text, as well as complex operations such as invoking methods on objects or modifying objects used by the rules. The rule structure is shown in Figure 1.

```
rule ruleName{
   priority = priorityValue;
   packet = packetName;
   property propertyName = value;
…
  when {conditions…}
  then {actions…}
  };
```

Figure 1: Rule Structure

Objects in JRules correspond to actual Java objects. To be evaluated by a rule, the object must exist in working memory. Placing an object in working memory is accomplished in JRules through the use of an **ASSERT** statement.

A sample rule for evaluating protocol compatibility is shown in Figure 2. This rule evaluates protocol compatibility between components that use RMI protocol version 1.1 and 1.0. In this example, if component **?c1** is implemented using RMI 1.1 and component **?c2** is implemented in the RMI 1.0 protocol, ten points are added to the compatibility score for the ensemble. This value is added because a well-defined interface exists between RMI version 1.1 and 1.0. Compatibility scores range from –10 (the lowest level) to 10 (the highest level).

There are two important things to do before we could execute these rules. First, the XML descriptions must be converted into Java objects. Then, these objects must be moved into working memory.

One approach for generating Java objects is to use the Java binding of the Document Object

```
rule ProtocolCompatible1{
  priority = high;
  when {
    ?c1: Component (protocol.equals("RMI1.1"); ?i1:id);
    ?c2: Component (protocol.equals("RMI1.0"); ?i2:id; ?i1 <?i2);
    ?e: Evaluation();
  }
  then {
    modify ?e{score+=10;}
  }
};
```

Figure 2: Protocol Compatible Rule

Model (DOM). This is a natural application of DOM since it is primarily intended as a document-to-object mapping tool. However, we did not attempt this approach, since we were concerned that the DOM objects did not provide the interface required by JRules. The fact that the collection of Java objects can be viewed as static, we simply translated a sample set of XML component specifications into Java classes. This translation is mechanical and should be straightforward to automate.

The problem of moving Java objects into working memory could not be solved easily, as only those objects returned by the XQL query were moved. We decided to use XSL Transformations (XSLT) (Clark, 1999) to generate the required ASSERT statements from the XML documents returned by the XQL query. These statements referenced the corresponding Java objects that had been generated statically beforehand.

Once these objects have been moved into working storage, the integration rules can be executed to rank the ensembles. Evaluation scores are accumulated for each ensemble set and the ensembles are presented to the system integrator in order of their rankings.

3. OVERALL SYSTEM STRUCTURES

The system consists of a searchable repository of component specifications, integration rules, query server, and component ensemble evaluator as shown in Figure 3. System integrators provide system

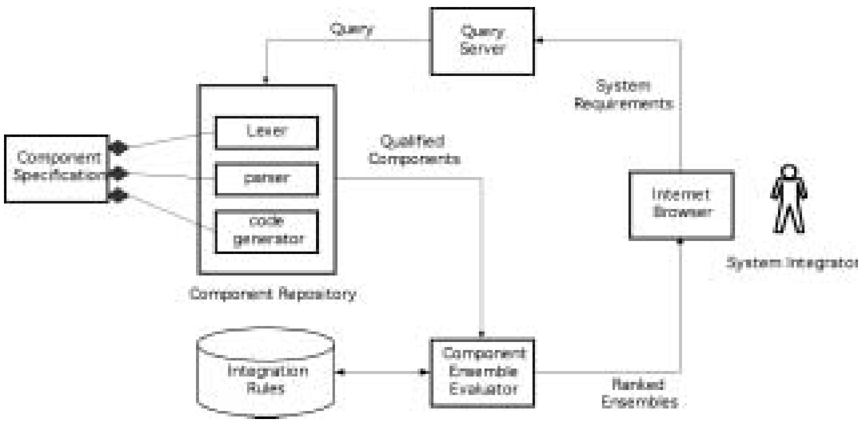


Figure 3: System Architecture

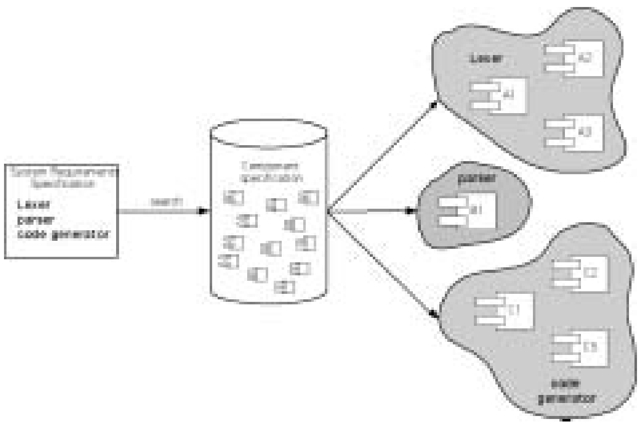


Figure 4: Qualified Components

requirements specifications that define the functional requirements of the components as well as any overriding constraints on how the components are integrated.

The requirements specification is converted to a series of queries on the component repository. Components that match the system requirements specification are grouped into ensembles.

Components in each ensemble are evaluated for compatibility based on the value of attributes in their component specification and a repository of software engineering integration rules. The attributes define characteristics that impact compatibility with other components, for example, the protocols supported by the component. Integration rules define how attributes affect component integration. These rules identify both those attribute combinations that simplify—and those that complicate—system integration.

The integration rule database may be extended by system integrators and component vendors. Integration rules typically reflect known compatibilities and incompatibilities between products. The discovery and refinement of these rules is a normal part of the system integration process. This makes it necessary to provide mechanisms for adding new integration rules to the database, and to modify and delete existing rules.

Finally, the ensembles, ranked according to compatibility, are returned to the system integrator for further evaluation.

4. DATABASE REPRESENTATION

4.1 Component Specification

XML (eXtensible Markup Language) is used in this experiment to represent both the component and system requirements specifications. XML is a World Wide Web Consortium (W3C) recommendation that has become universally accepted as the standard for document interchange (Bray *et al*, 1998). XML is well suited for this application as it provides a formal representation for mapping attributes to values and is fully extensible (Mundie, 1997; Cover, 2000). Figure 5 shows a part of the Document Type Definition (DTD) for a component specification.

The component specification contains general information, component interface information, and a component ID. Some attributes included in the component specification borrow from Poulin and Werkman (1995). The general information includes the component name, version, vendor, platform and functionality and information about the system interfaces. Functionality may be described

```
<?xml encoding=”UTF-8”?>
<!ELEMENT components(component+)>
<!ELEMENT component(general_info, protocol+, transaction_management?,security?)>
<!ATTLIST component id ID #REQUIRED>
<!ELEMENT general_info (name, version, vendor, platform, function+, framework,
language, sys_req, domain, keywords, gui?, cost, license, lang_supported)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT platform (#PCDATA)>
<!ELEMENT function (#PCDATA)>
…
<!ELEMENT protocol (name, version, provider, RMI_protocol?)>
<!ATTLIST protocol credential CDATA #REQUIRED>
<!ELEMENT provider (name, phone, address, url, contact+)>
…
<!ELEMENT transaction_management EMPTY>
…
<!ELEMENT security (confidentiality?, authentication?, non_repudiation?)>
<!ELEMENT (#PCDATA)>
```

Figure 5: Component Specification DTD

informally using key terms or with a formal specification language such as the one described by Mili *et al* (1997). Vendor information is used to furnish additional information about the component.

Component interfaces documented in the component specification are not simply the signatures of method or function calls in the component, but include non-functional properties such as performance, accuracy, availability, latency, and security. This expanded use of the term interfaces is described by Bachmann *et al* (2000) in their treatise on component-based software engineering.

ComponentID serves as a unique identifier of the component.

Prieto-Diaz and Freeman (1987) suggest that the characterisation of the functionality of a software component and its environment suffice for classification. We include information about functionality and environment in our component specification but include additional information regarding component characteristics that may affect integration with other components. Gorman (1999) states that the inclusion of additional attributes may improve the effectiveness of component search. It is important to justify the inclusion of these attributes in the component specification and not just add them extemporaneously. Poulin (1999) suggests that collecting large amounts of metadata to help retrieve components wastes time and money, and makes the library both difficult to contribute to and difficult to retrieve from. It is therefore necessary to limit the selection of attributes to those that have a significant impact on the suitability of a component for integration. The availability of language bindings, for example, is an important attribute that greatly influences the degree of difficulty involved in integrating a component.

4.2 System Requirements Specification

The system requirements specification is also represented in XML. The system requirements specification consists of one or more component specifications and a set of system constraints. In an actual system this specification may in fact be simply one element of a larger artifact.

System constraints use the same collection of attributes as individual components, for example, Java which may be specified as the language of choice for the system. Component specifications in the system requirements specification are normally sparsely populated—often the functional requirements alone are specified. This is because elements in the system requirements specification are treated as absolute constraints on the system (they are, after all, requirements). For example, if the platform is specified as Windows NT, only components that are available on this platform are identified. Attributes specified in the system requirements specification for individual components similarly limit component candidates to those that match the requirements. This produces a tension between search constraints and search results and, correspondingly, between requirements and available components. As search constraints are relaxed, additional but less-qualified components will be identified. On the contrary, as additional constraints are added, a smaller group of better-qualified components will be identified.

The ability to modify requirements and re-execute a search is a major benefit of automating this process—allowing system integrators to adjust system requirements to market realities in real time.

5. EXPERIMENT

To validate our approach we created a model problem consisting of a system requirements specification for a compiler consisting of a lexer, parser and code generator and a component repository. Figure 6 shows the random sampling of ensembles selected, ranked by score. These ensembles are selected from a component repository containing three lexical analysis tools (component IDs: 100, 115, 210), five parsers (component IDs: 105, 106, 107, 109, 1001) and two code generators (component IDs: 101, 103). There are total of 30 possible combinations (ensembles) of components in the repository that meet the system requirements specification.

Ensemble	Component IDs	Compatibility Score
21	115, 101, 107	39
20	210, 101, 107	39
8	210, 101, 106	29
26	210, 101, 105	29
24	115, 101, 105	29
3	115, 103, 106	20

Figure 6: Compatible Score Level of Ensemble Sets

Ensembles 20 and 21 both share a high score of 39. These scores result from the application of the integration rules to the set of attributes defined for the components included in each ensemble. Components in both of the highly ranked ensembles share a number of attributes, whilst the remaining attributes are not highly incompatible, resulting in relatively high overall scores for the ensembles.

6. FUTURE WORK

Until this model can be realised as an operational system, or at least scaled up significantly, it is difficult to provide a more compelling example. The first step in this process should be to define a "standard" component specification. Our current focus is to create a core set of common attributes that can then be extended for a specific component model and domains. Furthermore, we hope to provide a set of necessary attributes for the integration from real-world implementations. This information will be included into the resulting ranked ensembles to benefit later system integrators who have similar system requirements specification. Currently, we are in the process of gathering information from software developers in real-world because there is very little published experience and knowledge concerning component integration. We hope the collected qualitative measures on integration attempts from domain experts will be influential on component-based software development. We also plan to handle mismatch software components by extending integration rules to reduce incompatible possibility.

7. CONCLUSION

Our model can be thought of as an expert system for system integration. It allows system integrators to explore a broader component space than is possible using manual techniques, and quickly eliminates components that are overly difficult to integrate.

In addition to aiding in the evaluation of component ensembles, our model provides a mechanism for preserving, sharing, and re-using hard-won system integration knowledge. System integrators can use this information to identify compatible ensembles of components and actively expand upon it as additional insights into the rules that govern system integration emerge.

The greatest challenge in this model is not the feasibility of automating the process, which we have demonstrated, but the ability to collect the data necessary to drive the process. First, it is necessary to populate the component repositories with a sufficient number of component specifications to guarantee that the system requirements specification can be satisfied from the pool of available components. Second, it is necessary to identify, through successive rounds of refinement, the attributes that are used to describe each component and the set of system integration rules that are used to compute the compatibility of ensembles.

ACKNOWLEDGEMENTS

Thanks to Dr. Yunyong Teng-Amnuay and Assist. Prof. Dr. Peraphon Sophatsathit for their valuable comments and suggestions.

REFERENCES

BACHMANN *et al* (2000): Technical concepts of component-based software engineering *Volume II (CMU/SEI-2000-TR-008, ADA379930)*. Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University.

BASILI, V.R. and BOEHM, B. (2001): COTS-based systems top 10 list. *IEEE Computer*, 34(5).

BRAY, T. *et al* (1998): Extensible markup language (XML) 1.0. *W3C Recommendation*, February 10.

CLARK, J. (1999): XSL transformations (XSLT) version 1.0. *W3C Recommendation*, November 16.

COVER, R. (2000): Literate programming with SGML and XML. *The XML Cover Pages*. http://www.oasis-open.org/cover /xmlLitProg.html, visited date December 20.

GORMAN, M. (1999): Metadata or cataloguing? A false choice. *Journal of Internet Cataloging* 2(1).

HARANDI, M. *et al* (1986): Rule base management using meta knowledge. *ACM SIGMOD* Record, Proceedings of the conference on Management of data, 15(2) June.

ILOG (2000): company home page, *http://www.ilog.com*, visited date: September 2000.

MILI, R. *et al* (1997): Storing and retrieving software components: A refinement based system. *IEEE Transaction on Software Engineering* 23(7) July.

MUNDIE, D. (1997): Standardized data representations for software testing. *Pacific Northwest Conference on Software Quality*, Portland, Maine, October.

ORSO, A. *et al* (1999): Component metadata for software engineering tasks. proceedings of *EDO 2000*, LNCS Vol. Springer.

POULIN, J. S. (1999): Reuse: Been there, done that. *Communication of the ACM* 42(5): 98–100.

POULIN, J. S. and WERKMAN, K. J. (1995): Melding structured abstracts and the world wide web for retrieval of reusable components. *Proceedings of the 17th International Conference, Symposium on Software Reusability on Software Engineering*, Seattle, Wash.: April 23–30.

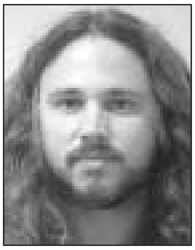
PRIETO-DIAZ, R. and FREEMAN, P. (1987): Classifying software for reusability. *IEEE Software* 4(1):6–16.

BIOGRAPHICAL NOTES

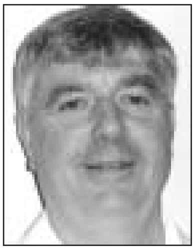
Somjai Boonsiri is an assistant professor at the Faculty of Science, Chulalongkorn University, Bangkok, Thailand. Her research interests are component-based software engineering, distributed technologies and database technologies. She has worked with Robert Seacord and David Mundie for the past two years at the SEI, Carnegie Mellon University as a visiting scientist. She has six years of system analysis and design experience in government organisations. Somjai has taught in universities for more than 10 years.

Somjai Boonsiri

Robert C. Seacord is a senior member of the technical staff at the SEI: Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, U.S.A. Seacord has 19 years of software development experience in industry, defence and research. Seacord's principal areas of expertise include component-based development, graphical interface design, human factors. He has worked extensively with EJB, CORBA, JavaBeans, UNIX, Motif, the Common Desktop Environment (CDE), and other graphical user interface systems and technologies. He is a member of the COTS-Based Systems (CBS) initiative. In 2001, he completed an SEI Series book Building Systems from Commercial Components *published by Addison Wesley with co-authors Kurt Wallnau and Scott Hissam.*


Robert Seacord

David A. Mundie is a senior member of the technical staff in the Networked Systems Survivability Program at the Software Engineering Institute (SEI). The CERT® Coordination Center is a part of this program. He is a member of the Institute of Electrical and Electronics Engineers, Inc. (IEEE) Computer Society. Mundie is adjunct professor at the Department of Information Science and Telecommunications, University of Pittsburgh, Pennsylvania. He is an orginator of CyberDewey: A catalogue for the World Wide Web.


David Mundie