Component Specification and Wrapper/Glue Code Generation with Two-Level Grammar using Domain Specific Knowledge *

Fei Cao[1], Barrett R. Bryant[1], Rajeev R. Raje[2], Mikhail Auguston[3], Andrew M. Olson[2], and Carol C. Burt[1]

[1] Department of Computer and Information Sciences
University of Alabama at Birmingham, USA
{caof, bryant, cburt}@cis.uab.edu

[2] Department of Computer and Information Science
Indiana University Purdue University at Indianapolis, USA
{rraje, aolson}@cs.iupui.edu

[3] Computer Science Department
New Mexico State University, USA
mikau@cs.nmsu.edu

Abstract. UniFrame is a framework for seamlessly assembling heterogeneous distributed components. It's based on the Unified Meta-component Model (UMM) for describing components. Systems constructed by component composition should meet both functional and non-functional requirements such as the Quality of Service (QoS). We propose a Component Description Language (CDL) to specify the UMM components based on domain specific knowledge in the context of UniFrame using Two-Level Grammar (TLG). CDL is also used for wrapper/glue code generation. A simple case study is illustrated to show how CDL may be applied.

Keywords: Two-Level Grammar, Wrapper/Glue Code, Domain Specific Knowledge, Formal Specification

1 Introduction

Current software environments feature heterogeneous platforms, languages and applications over distributed systems. The trend of integrating software systems is already on the horizon. UniFrame [1] is a framework for seamless interoperation of heterogeneous distributed software components. It's based on the

* This material is based upon work supported by, or in part by, the U. S. Office of Naval Research under the award number N00014-01-1-0746, and the U. S. Army Research Laboratory and the U. S. Army Research Office under the contract/grant number DAAD19-00-1-0350.

Unified Meta-component Model (UMM) [2] for describing components. Systems constructed by component composition should meet both functional and non-functional requirements such as the Quality of Service (QoS) [3]. A Generative Domain Model (GDM) [4] is used to describe the properties of domain specific components and to elicit the rules for component assembly. In UMM, an infrastructure called "Headhunter" is proposed to actively detect the presence of new components in the search space, register their functionality and attempt matchmaking between client components (service requesters) and server components (service providers). As such, our specification should incorporate the mechanism of the Headhunter [5] to represent component information for retrieval and subsequent generative assembly. Here we adopt Gruber's view of ontology [6] to make the conceptualization of a domain explicit, and perform specification matching within a domain context using domain specific knowledge.

In this paper, we propose a Component Description Language (CDL) using Two-Level Grammar (TLG) [8] as the formalism not only to represent the UMM, QoS, and GDM, but also for wrapper/glue code generation in the component assembly process. We begin by briefly introducing the concept of TLG (section 2) and then apply the TLG into the context of component specification by describing our CDL over a simple banking example (section 3).

2 Specification with Two-Level Grammar

TLG was originally developed as a specification language for programming language syntax and semantics [7]. The name "two-level" of TLG comes from the fact that TLG contains two context-free grammars corresponding to the set of type domains and the set of function definitions operating on those domains. Though based on natural language, TLG is also a formal notation in the sense that it's strictly typed. Its semantics are embodied by rules in the function bodies. We will illustrate this in the CDL example below. For a detailed explanation of TLG, please see [8].

In this paper, we describe a CDL as a subset of TLG to describe both UMM components and generative rules for component composition with the output of wrapper/glue code. This includes two levels:

- User level
  Based on UMM, CDL specifies the following items, namely Computational Attributes, which include Functional Attributes such as Syntactic Contract and Implementation Technology; Cooperative Attributes; Auxiliary Attributes; and QoS. Examples of QoS metrics include Security, Availability, Throughput. Please refer to [3] for details. User-level specifications are provided by component developers.
- System level
  This level mainly specifies the rules of component composition. Specifications of this level are to be used by component assemblers. The above statements are illustrated in Figure 1.

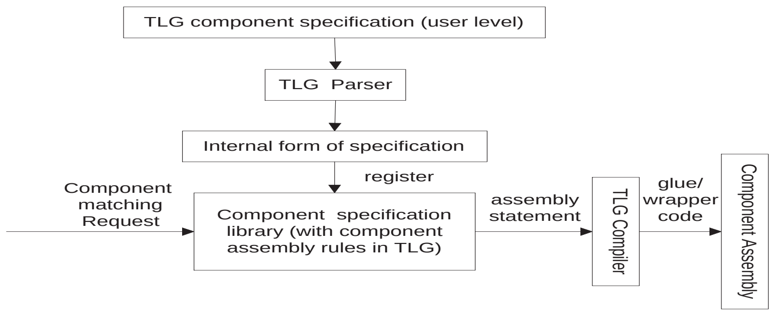
TLG component specification (user level)
TLG Parser
Internal form of specification
register
Component matching Request
Component specification library (with component assembly rules in TLG)
assembly statement
TLG Compiler
glue/ wrapper code
Component Assembly

Fig. 1. Application of TLG for Component Assembly

3 An Example

Consider a CORBA component model for a simple banking system. The simple bank offers three services: withdrawing money, checking account balance and making deposits.

3.1 User Level Component Model Specification

For simplicity, assume this simple bank component uses string-form Interoperable Object Reference (IOR) for locating objects as an alternative to the Directory or Naming service [9]. Assume UMMComponent is the root specification of all components specified in the component repository.

```
component SimpleBank extends UMMComponent.
  Technology:: String.
  Domain:: String.
  IOR:: String .
  Amount :: Float.
  AccountNumber :: String.
  Pin :: String.
  Withdraw:: FunctionName.
  Balance:: FunctionName.
  Deposit:: FunctionName.
```

The specification above defines the type domain for the SimpleBank component. The domain variables all begin with an upper case letter, so do component names, function parameters and type identifiers. The following are the function definitions operating on those domains:

```
  init: Domain ::= "Bank", Technology::="CORBA",
    IOR::="IOR:000000saji98898dss3322".
  export Withdraw with Amount AccountNumber Pin
    by classAccount returnType Boolean.
  export Balance with AccountNumber Pin by classAccount
    returnType Float.
  export Deposit with Amount AccountNumber Pin by classAccount
    returnType Boolean.
  /* Bank Domain Lexicon */
  lexicon of Withdraw: "draw";"subtract".
  lexicon of Balance : "check balance".
  lexicon of Deposit : "credit";"save".
  /* static Quality of Service */
  qos availability: 90%. //the duration that a component is available
  qos delay: 10ms./*the time between service invocation and completion */
end component SimpleBank.
```

init is the initialization routine for the type domain. Note that the function body on the right side of ':' specifies the rules of the left hand side function signature, as you may see from the "ComponentRepository" specification in the next section. There may be no rules at all, like in the above example.

The *export* expressions indicate the syntactic contract specifications of components. They represent the services offered by components. Moreover, the precondition of service offering may be specified in the function body. Note "classAccount" represents a component (in the form of class) named "Account" that provides the above three services; "=" is a relational operator while "::=" is used to set values.

From the above example we can see that specification matching can be carried out by looking for such function signatures as directed by "export", "qos", "technology", etc. "Domain" is also a crucial tag that uniquely determines the context in which the domain knowledge may be applied against this component. By "lexicon of" we provide a thesaurus for a specific syntactic contract, so that "withdraw" is treated as equivalent service to "draw", while "balance" is treated as an equivalent service to "check balance", etc. During the process of component retrieval and assembly, if a component with expected service offerings doesn't satisfy the expected QoS, it is subject to be substituted for.

3.2 System Level Component Assembly Specification

In the following "ComponentRepository" specification, code generation rule is specified.

```
UMM ComponentRepository
ComponentN::UMMComponent.
OperationName:: String.
Package:: StringList.//the header files to be included/imported
generate java wrapper code ComponentN
  OperationName using CORBA with Package :
  ComponentN get OperationName!= Empty,
  ComponentN init,
  ComponentN get Technology = "CORBA",
  ComponentN get IOR!= Empty,
  return CorbaClientCode.
end UMM ComponentRepository
```

The above statement specifies wrapper code generation for a CORBA component to be used by a Java client. **return** *CorbaClientCode* directs the TLG compiler to generate the actual CORBA client code. Here the semantics for **generate java wrapper code** are based on CORBA's Dynamic Invocation Interface [9], which lets a client pick any target object at run time and dynamically invoke its method without requiring precompiled stubs. Because of the limited space, the code generation process and detailed generated code are not illustrated here - see [10] for more details.

4 Conclusion

In this paper we use Two-Level Grammar as a formalism to represent the Unified Meta-component Model of a simple banking domain example and to generate the wrapper/glue code based on the generative domain model. We illustrate some empirical techniques on building a Component Description Language using TLG and evaluate how TLG may be leveraged toward this goal.

References

1. R. R. Raje, B. R. Bryant, M. Auguston, A. M. Olson, C. C. Burt: A Unified Approach for the Integration of Distributed Heterogeneous Software Components. Proc. 2001 Monterey Workshop Engineering Automation for Software Intensive System Integration, 2001, pp. 109-119.
2. R. R. Raje: UMM: Unified Meta-object Model for Open Distributed Systems. Proc. ICA3PP 2000, 4th IEEE Int. Conf. Algorithms and Architecture for Parallel Processing, 2000.
3. G. J. Brahnmath, R. R. Raje, A. M. Olson, M. Auguston, B. R. Bryant, C. C. Burt: A Quality of Service Catalog for Software Components. Proc. $(SE)^2$ 2002, the Southeastern Software Engineering Conf. (to appear), 2002.
4. K. Czarnecki, U. W. Eisenecker: Generative Programming, Methods, Tools, and Applications. Addison Wesley, 2000.
5. N. N. Siram, R. R. Raje, A. M. Olson, B. R. Bryant, C. C. Burt, M. Auguston: An Architecture for the UniFrame Resource Discovery Service. Proc. 3rd Int. Workshop Software Engineering Middleware (to appear), 2002.
6. T. R. Gruber: A translation approach to portable ontology specifications. Knowledge Acquisition, 5(2), 1993, pp. 199-220.
7. A. van Wijingaarden: Revised report on the algorithmic language ALGOL 68. Acta Informatica, 5, 1974, pp. 1-236.
8. B. R. Bryant, B.-S. Lee: Two-Level Grammar as an Object-Oriented Requirements Specification Language. Proc. 35th Hawaii Int. Conf. System Sciences, 2002, `http://www.hicss.hawaii.edu/HICSS_35/HICSSpapers/PDFdocuments/STDSL01.pdf`.
9. R. Orfali, D. Harkey: Client/Server Programming with Java and Corba. Wiley, 1998.
10. B. R. Bryant, M. Auguston, R. R. Raje, C. C. Burt, and A. M. Olson: Formal Specification of Generative Component Assembly Using Two-Level Grammar. Proc. SEKE 2002, 14th Int. Conf. Software Engineering and Knowledge Engineering, 2002, pp. 209-212.