Generative Programming Using Two-Level Grammar in UniFrame[*]

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

The UniFrame project consists of a Unified Meta-component Model (UMM) for distributed component-based systems (DCS), and a Unified Approach (UA) for integrating heterogeneous components [6]. It is assumed that different developers will provide on a network a variety of possibly heterogeneous components for specific problem domains and the UMM specifications of these components, which describe both their functionality and Quality of Service (QoS). QoS requirements are expressed in terms of an appropriate set of parameters selected from a catalog of parameters [2]. For a specific problem, a search process, employing components called "headhunters," identifies sets of components that satisfy both the desired functionality and QoS from those available on the network [7]. After the component sets are identified and fetched, the distributed application is assembled by choosing one component from each set according to the generation rules embedded in a generative domain-specific model (GDM) [4] supporting component-based system assembly. This assembly may require the creation of a glue/wrapper interface between various components as well as instrumentation to validate dynamic QoS parameters (e.g., response time). Two-Level Grammar (TLG) [3] is used to specify the generative rules and provide the formal framework for the generation of the glue/wrapper code. Event grammars [1] are used to produce the QoS dynamic metrics to be measured and validated at run-time. Once the system is assembled, it must be tested using a set of test cases supplied by the system developers to verify that it meets the desired QoS criteria. If it does not, it is discarded. After that, another implementation is chosen from the collection of component sets. This process is repeated until an optimal (with respect to the QoS) implementation is found, or until the collection is exhausted. In the latter case, the process may request additional components or it may attempt to refine the query by adding more information about the desired solution from the problem domain. If a satisfactory implementation is found, it is ready for deployment.

An overview of this approach has been presented in [8]. Here we concentrate on the generative programming language issues raised by using Two-Level Grammar (TLG). The two levels of the grammar are type definitions in the form of a context-free grammar, and function definitions in the form of another context-free grammar. These type and function definitions are encapsulated in a class and their combined functionality is equivalent to that of any other formal specification language. TLG may be used to provide for attribute evaluation and transformation, syntax and semantics processing of languages, parsing, and code generation. It is implemented through translation into VDM++, the object-oriented extension of the Vienna Development Method, which may further be translated into UML, Java, or C++ [5]. TLG is used to both formally specify aspects of components that facilitate their location by headhunters, as well as provide a notation for expressing the generative domain models and associated generative rules for integrating heterogeneous components.

Specification of components in TLG follows the framework established by UMM. The component interface has three aspects: lexical, syntactic, and semantic. The lexical aspect is the naming of the various functions, the syntactic aspect is comprised of the function signatures, and the semantic aspect is

[*] This material is based upon work supported by, or in part by, the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant numbers DAAD19-00-1-0350 and 40473-MA, and by the U. S. Office of Naval Research under award number N00014-01-1-0746.

[1] Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL 35294-1170, U. S. A., {*bryant, caof, zhaow, cburt*}*@cis.uab.edu.*
[2] Department of Computer and Information Science, Indiana University Purdue University Indianapolis, Indianapolis, IN 46202, U. S. A., {*rraje, aolson*}*@cs.iupui.edu.*
[3] Computer Science Department, New Mexico State University, Las Cruces, NM 88003, U. S. A., *mikau@cs.nmsu.edu.*

the functionality and QoS of the functions. A server may use a different name for a service it provides than a client will use when requesting that service. We use a domain knowledge-base augmented with a natural language lexicon to perform the lexical matching of names. The signatures may also vary slightly as some function parameters may not be needed or may be replaced with default values. However, closely matching signatures may clarify uncertain matches at the lexical level. Semantic specification further clarifies mismatches. Semantically consistent functions would have comparable pre-conditions and post-conditions, affect comparable QoS parameters, etc.

Our case study is a bank account management system. Assume that different client and server components for a bank domain are available on the network. These components (belonging to a category, i.e., server/client) offer the same functionality, perhaps with different lexical and syntactic descriptions, but different QoS features. Let the headhunters find the following components: **JavaAccountClient**, **JavaAccountServer**, and **CorbaAccountServer**, with the first two adhering to the Java-RMI model and the third one developed with CORBA technology. Suppose that the UMM specifications associated with the components indicate that the two server components have the same functionality but **CorbaAccountServer** has better service guarantees and meets the QoS specified in the system query. Thus, the final system should be assembled from the Java client and the CORBA server. Based on the generation rules embedded in GDM, a proxy server for the Java client component, a proxy client for the CORBA server component, a bridge driver between the two proxies and some other installation helper files can be generated to form an integrated account system. The assembled system will be deployed if it meets the desired QoS criteria. To effect this deployment, a new set of UMM specifications will be generated for the integrated system to insure that it is available for discovery by other head-hunters, i.e., to act as a component of other possible application systems.

References:

[1] M. Auguston, A. Gates, M. Lujan, "Defining a Program Behavior Model for Dynamic Analyzers," *Proc. SEKE '97, 9th Int. Conf. Software Eng. Knowledge Eng.,* 1997, pp. 257-262.

[2] G. J. Brahnmath, R. R. Raje, A. M. Olson, M. Auguston, B. R. Bryant, C. C. Burt, "A Quality of Service Catolog for Software Components," *Proc. Southeastern Software Eng. Conf.* (to appear), 2002.

[3] B. R. Bryant, B.-S. Lee, "Two–Level Grammar as an Object-Oriented Requirements Specification Language," *Proc. 35th Hawaii Int. Conf. System Sciences*, 2002.

[4] K. Czarnecki, U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley, 2000.

[5] IFAD, *The VDM++ Toolbox User Manual*, 2000. http://www.ifad.dk

[6] R. R. Raje, M. Auguston, B. R. Bryant, A. M. Olson, C. C. Burt, "A Unified Approach for the Integration of Distributed Heterogeneous Software Components," *Proc. Monterey Workshop Engineering Automation for Software Intensive System Integration*, 2001, pp.109-119.

[7] N. N. Siram, R. R. Raje, B. R. Bryant, A. M. Olson, M. Auguston, C. C. Burt, "An Architecture for the UniFrame Resource Discovery Service," *Proc. SEM 2002, 3rd Int. Workshop Software Engineering and Middleware* (to appear), 2002.

[8] W. Zhao, B. R. Bryant, R. R. Raje, M. Auguston, A. M. Olson, C. C. Burt, "A Unified Approach to Component Assembly Based on Generative Programming," *Proc. GP 2002, 2002 Generative Programming Workshop*, 2002. http://www.cwi.nl/events/2002/GP2002/papers/zhao.pdf