A Component Assembly Architecture with Two-Level Grammar Infrastructure[1]

Wei Zhao[2] Barrett R. Bryant[2] Fei Cao[2] Rajeev R. Raje[3] Mikhail Auguston[4]
Andrew M. Olson[3] Carol C. Burt[2]

1. Introduction

Being able to generate a concrete software product from domain specifications, upon an order requirement[5], still remains a mirage using most modern software engineering techniques. To provide a systematic way to automate software engineering process, formal models should be constructed beforehand to capture the various aspects of engineering knowledge for any predictable software solutions for a particular domain; an infrastructure should be available to support the automation of any specific product generation by intelligently using the established engineering knowledge models.

Engineering knowledge involves the decisions made about a software product along its production line, which includes the policies from domain business executives, expertise from domain experts, experiences from software managers and engineers, and the techniques from software developers and programmers. During the software production process, these engineering knowledge will contribute respectively towards service specifications of the system and the Quality of Services (QoS), detailed business logic of the system, specifications of software architecture and role assignments for developers, concrete software development by applying different programming languages and component-based technologies.

Using current software engineering practices, the investments of engineering knowledge are all encapsulated in one business organization, making engineering knowledge implicit, vague and intertwined. However, the latter two aspects are from technology prospective other than business prospective, and can be most possibly reused across all the business domains. To construct formal models that capture various aspects of engineering knowledge, and to organize them in such a way that separation of concern and maximized reuse of engineering knowledge can be achieved, we categorize this synergy of engineering knowledge into three-dimensional domains:

1) Business domains are associated with the natural categorization of business sectors and the natural hierarchical structure of business organizations;
2) Functionality domains are based on the functionality and the role of different parts of software, and their collaboration means and patterns; and
3) Technology domains address the issues related to software implementation technologies such as component models, programming languages, hardware platforms, and so on.

Different group of people or organizations are expected to be responsible for each domain. The successful construction of the Generative Domain Model (GDM) [Cza00] (for each domain

[1] 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.
[2] Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL 35294-1170, U. S. A., {zhaow, bryant, caof, cburt}@cis.uab.edu.
[3] Department of Computer and Information Science, Indiana University Purdue University Indianapolis, Indianapolis, IN 46202, U. S. A., {rraje, aolson}@cs.iupui.edu.
[4] Computer Science Department, New Mexico State University, Las Cruces, NM 88003, U. S. A., mikau@cs.nmsu.edu.
[5] By "order requirement", we mean either a requirement document written in natural language by human or a system construction request from another computer program.

mentioned above), would assist in automating the development of software products under the guidance of model transformations and refinements from the highest model (GDM) to more specific intermediate models. This would finally lead to the end software products. This paper describes the UniFrame project that aims at this goal.

2. Related Work

Toward the goal of automatic production of software, there have been many attempts in domain engineering, system generation and model transformations. We describe a few prominent ones here.

Generative Programming [Cza00] is well known for providing a vision of automatically generating products from a GDM, a specification of the product domain. However, the examples provided for elementary components envisioned by the authors are limited to C++ **structs** and **classes** with templates, which may not be sufficient to solve problems on the scale of distributed and component-oriented computing. Many problems like universal interoperability should be solved during system integration and generation. Widely known efforts such as CORBA [Corba], Web Services [W3C] and Model Driven Architecture (MDA) [OMG01], an initiative of the Object Management Group (OMG), arose as possible solutions for the interoperability problem.

MDA sketches out a model transformation series, which transforms a business model to a Platform Independent Model (PIM), then to a Platform Specific Model (PSM), and finally gets to the executable code. Steps of model transformations certainly contribute to the automated product generation from the high level specifications. Nevertheless, MDA currently appears to be only concentrating on the model transformation for a single system. It also does not specify the assembly of a system out of many available components.

FORM [Kan98] provides methods to construct feature models for a domain during the domain engineering phase and to generate concrete systems by applying feature selection during the application engineering phase. FORM defines domain features in terms of services, domain technologies, operating environments and implementation techniques. We do consider it to be inappropriate that the feature models for a business domain should include the latter two, as it is not a good practice of separation of concerns, and can be a further obstacle for system flexibility evolvability and engineering knowledge reuse. The architecture defined in FORM from three different viewpoints (subsystem, process and module) does not capture the multi-dimensions of engineering knowledge during a product manufacturing process.

3. UniFrame Architecture Overview

The UniFrame project[6] is a framework for:

1) Providing an architecture for automated software product generation, upon an order requirement, based on the assembly of a selection from an ensemble of searched components (with which we believe we can overcome limitations mentioned in section 2);
2) Providing a practical technique based on the formalism of Two-Level Grammar (TLG) [Bry02], which serves as the infrastructure enabling the automation of software production by steps of model transformations.

UniFrame has two levels:

[6] This project goal statement is phased according to our newly developed ideas and is different from the original official one, which is "Seamless integration of heterogeneous and distributed software components" [UniFr].

- GDMs for business domain, functionality domain and technology domain jointly comprise the core part of the system level of UniFrame. The GDM for the business domain mainly contains: domain feature models, standardized elementary domain service[7] specifications uniquely identified by their Universal Resource Identifiers (URIs), associated Quality of Service (QoS) parameters, service collaboration patterns, typical computing algorithms for this business domain, domain specific language, etc. Some preamble of a business GDM may be a standardized **Stack** class provided by J2SE [J2SE] for the domain of the object "stack", or OpenGL [OpenG] for the domain of graphics and images processing. The functionality domain GDM is essentially a reference architecture model that identifies the functionality, the role and the collaboration patterns among different parts of software. The GDM for technology domain deals with the interoperability across heterogeneous implementation technologies and programming languages. The UniFrame system level sets the context for developing a family of products. We propose an Internet Component Exchange and Assembly (ICEA)[8] center for each business domain for developing and maintaining the business GDM.
- The UniFrame component level gives the view of the single system development. Component developers have the freedom of choosing any implementation technology, underlying hardware platform, or programming language to implement any standardized service or a group of services confining to the service specifications in the business GDM. The developer even has the freedom to name services as long as a DNS server (specialized in this context) can perform the correct translation to the one with standardized semantics and unique URI in the business GDM. Upon the accomplishment of the individual component development, developers need to fill out a Unified Meta-Component Model (UMM) [Raj01] form to formally describe the components. UMM identifies the niches of this component in various GDMs, provides the QoS of this component and the address of the native component registry (e.g. RMI registry if this component is developed in RMI). Then the developers need to register the UMM to its respective ICEA. Hopefully, in the future, this process can be further facilitated by MDA techniques: the developers pick up the business model for any business services, and apply the model transformations to get to the executable code.

TLG is used to represent the three GDMs and the UMM. Because the domain services are standardized and factored, it is feasible for the users to explicitly identify the service semantics in their order requirements. The automatic production is carried out by the joint-effort among the feature models in the three GDMs, feature selection from order requirements, and feature identification and concretization in the UMM. At the system generation time, we need apply the service interaction patterns from the feature models in business GDM for homogeneous components; if the components are heterogeneous, we need apply the component interaction patterns from the functionality GDM, and then use the mapping and translation rules stored in the technology GDM for building interoperability. More precisely for interoperation, the UMM specification (in TLG) will be translated into WSDL [W3C][Cao02], making Web Services the underlying communication technique. The model transformation computation supporting product automation is performed by the TLG interpreter that computes steps of substitution (the first level context-free grammar) between two models (grammar's left and right hand side) guided by the transformation rules (the second-level context-free grammar). Different levels of models will be

[7] The "service" is not an executable entity. It is a concept of a slot of domain businesses. The "component" defined under a component model can be executed within a component framework. The component developers build software components by concretizing services. The "component" is a technologic carrier for "services".
[8] It is our notion of a group of people or organizations for this purpose.

represented by groups of TLG classes, e.g. Class Withdraw is a service description in the bank domain GDM.

```
class Withdraw.
   Passin :: AccountNumber, Amount.
   …..
end class.
```

A lower level model could further define AccountNumber as:

```
class AccountNumber.
     Type :: Integer.
     Language :: c++.
     …..
end class.
```

Or as:

```
class AccountNumber.
     Type :: String.
     Language :: java.
     Lexeme :: letter (letter | number )*.
     …..
end class.
```

Please refer to [Bry02] for more details on TLG, and refer to [Zha02] for our current definition and examples of TLG as an executable code generator.

4. Engineering Principles Employed in Designing UniFrame

Various engineering principles are observed in designing UniFrame architecture to fulfill its goal:

- Modularity is the fundamental consideration in designing UniFrame. In UniFrame, the final system (product) is built from components, which in turn are built around one or more services. The atomic and factored services (or features) is the truth that the system can be **generated** on demand from requirements, in another word, across all the products of a product family, what can really be reused and re-structured are the elementary services. Given all the possible elementary services for a business domain, a wide spectrum of systems can be generated by various combinations of services. Service composition rules (e.g. domain feature models) are embedded in the business GDM, and the component composition rules [Sun02] are embedded in the functionality GDM.
- The principle of autonomy and separation of concerns naturally separates the multidimensional engineering knowledge into three GDMs maintained by different groups of people, respectively. On the maturity of UniFrame, we hope the stabilized infrastructure will have three sets of APIs that will enable the creation/maintenance of these three GDMs. The experts in different domains have the freedom of controlling their domains; the component developers have their own choice about the implementation details. This makes UniFrame flexible, dynamically re-configurable and evolvable.
- UniFrame also supports a transparent communication channel. The business GDM with standardized services and their QoS is the communication media among the users, the system and the component developers, which ensures what the component developer supplies and what the system produces is exactly what the users want. It also suggests that the automated production could start from as early as order requirements.

- Reflection and intelligent reasoning of model transformations with minimum human interaction is also a key attribute of UniFrame. UMM, a reflection of a component, together with three GDMs provide the TLG-facilitated infrastructure enough knowledge to pursue intelligent reasoning in the process of system assembly, e.g. automatically reason about component properties and relationships.

5. Two-Level Grammar

As UniFrame maturates, the infrastructure is not intended for frequent human manipulations. It is reasonable to choose TLG (textual with functional and logic programming language style) as the machine-understandable infrastructure and use UML as the human-system interface (e.g., for representing GDMs and transformation rules externally). Tools will be constructed to perform the translation between internal and external representations.

With natural language-like syntax, a TLG specification is self-descriptive and very understandable. Therefore, TLG has more potential to be mastered by software engineers than other formal methods such as Z [Spi89].

XML is very suitable for data exchange and description, but not for code generation or even more complicated tasks like model transformations. In a pure sense, XML carries no more semantic meaning than HTML. XML itself does not perform a computation, but relies on the intelligence of non-reusable XML processing engines. On the other hand, TLG is Turing complete with very nice logic and functional language style reasoning. Regarding readability, the frequent use of angle bracket templates in Xpath and XSLT [Cle01] makes the readability of the generator poor. TLG offers improvements in readability, as well.

TLG is Object-Oriented (OO), making it a good candidate for formal specification of OO computing entities. Additionally, TLG goes beyond OO programming languages with its unique syntax and semantics. A simple rule such as:

NewObject:: {Object1}* Object2, Object3; Object4.

states a rather complicated feature selection and federated construction of the NewObject: the NewObject could be constructed by zero or more instances from domain Object1 followed by an instance from domain Object2, and an instance from domain Object3; or the NewObject could be constructed by an instance from domain Object4. It would require a large block of statements in an object-oriented programming language to represent the same intent. In TLG, it is very easy to combine objects and flat entities (literals) together as features because both terminals and nonterminals are allowed on the right hand side of meta-rules.

TLG has two levels. The meta level computation can be viewed as model/pattern transformations. More abstract patterns on the left hand side can be substituted by many combinations and alternatives of more specific patterns on the right hand side of the grammar. The hyper level context-free grammar (together with the consistent substitution) sets the context for the first one: rules and logic for applying patterns, very suitable for plug-and-play component composition. Also for each context-free grammar, we can automate the feature configuration validation and constraint checking [Jon02], leveraging widely available open parser and type checker generator facilities such as CUP [CUP99].

6. Conclusion

This extended abstract provides the overview of the UniFrame architecture, considerations in designing UniFrame and the issues of infrastructure implementations. The novel contribution of UniFrame is to bridge the gap between the vision of Generative Programming and the existing

MDA framework: we provide a practical architecture and a infrastructure technique using the MDA model transformation idea to fulfill the goal of Generative Programming.

7. References

[Bry02] B. R. Bryant, B.-S. Lee, "Two–Level Grammar as an Object-Oriented Requirements Specification Language," *Proc. 35th Hawaii Int. Conf. System Sciences*, 2002.

[Cao02] Fei Cao, Barrett Bryant, Carol Burt, Rajeev Raje, Mikhail Auguston, Andrew Olson. "A Translation Approach to Component Specification," (poster), OOPSLA'02,2002.

[Cle01] J. C. Cleaveland. *Program Generators with XML and JAVA*. Prentice Hall 2001.

[Corba] Common Object Request Broker Architecture (CORBA), http://www.corba.org/

[CUP99]CUP parser generator for Java. http://www.cs.princeton.edu/~appel/modern/java/CUP/

[Cza00] Czarnecki, K., Eisenecker, U. W., *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[J2SE] Java™ 2 Platform, Standard Edition, http://java.sun.com/docs/index.html

[Jon02] M. D. Jonge, J. Visser "Grammars as Feature Diagrams" P*roceedings of Workshop on Generative Programming,* April 2002. http://www.cwi.nl/events/2002/GP2002/papers/dejonge.pdf

[Kan98] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, Moonhang Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Annals of Software Engineering* 5, pp. 143-168, 1998.

[OMG01] Object Management Group. Model Driven Architecture: A Technical Perspective. Technical Report. Document #ormsc/2001-07-01. Framingham, MA: Object Management Group. July 2001.

[OpenG]OpenGL. http://www.opengl.org/

[Raj01] 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. 2001 Monterey Workshop Engineering Automation for Software Intensive System Integration*, 2001, pp. 109-119.

[Sun02] C. Sun, R. R. Raje, A. M. Olson, B. R. Bryant, M. Auguston, C. C. Burt, Z. Huang, "Composition and Decomposition of Quality of Service Parameters in Distributed Component-Based Systems," to appear in *Proc. Fifth IEEE Int. Conf. Algorithms and Architectures for Parallel Processing*, 2002.

[Spi89] J. M. Spivey, The Z notation: a reference manual. Prentice Hall, New York, 1989.

[UniFr] UniFrame http://www.cs.iupui.edu/uniFrame/

[W3C] World Wide Web Consortium, Web Services, http://www.w3.org/2002/ws/

[Zha02] W. Zhao. "Two-Level Grammar as the Formalism for Middleware Generation in Internet Component Broker Organizations." *Proceedings of GCSE/SAIG Young Researchers Workshop*, held in conjunction with the *First ACM SIGPLAN Conference on Generative Programming and Component Engineering*, 2002. http://www.cs.uni-essen.de/dawis/conferences/GCSE_SAIG_YRW2002/submissions/final/Zhao.pdf