A Language Based Formalism for Domain Driven Development

Wei Zhao
Computer and Information Sciences, University of Alabama at Birmingham
Birmingham, AL 35294-1170, USA
1-205-934-2213
zhaow@cis.uab.edu

ABSTRACT
The evolution of programming languages (e.g. machine languages, assembly languages and high level languages) has been the driving force for the evolution of software development from the machine-centric to the application-centric. The 4th generation languages (4GLs), languages defined directly by the composition of domain features, serve as the language-based formalism for the emerging Domain Driven Development paradigm. The 4GLs are defined in Two-Level Grammar++ and can be compiled into 3GLs using the 4GL compiler framework.

Categories and Subject Descriptors
D.2.13 [**Software Engineering**]: Reusable Software – *domain engineering.*

D.3.1 [**Programming Languages**]: Formal Definitions and Theory – *semantics, syntax.*

F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems – *Grammar types.*

General Terms
Languages, Theory, Standardization, Reliability.

Keywords
4th Generation Languages, Feature Model, Generative Domain Model, Two-Level Grammar, Domain Engineering, Application Engineering, 4Compiler.

1. INTRODUCTION
Domain Driven Development (3D) covers many related research efforts such as Generative Programming (GP) [2], Product-line Architecture, Feature-Oriented Programming [1], Domain-Specific Languages (DSLs) [3], Domain-Specific Modeling [5], and Model-Driven Architecture (MDA) [4]. The essential goals of these technologies are 1) moving the development abstraction up toward the domain, 2) achieving higher automation in software development, and 3) achieving a higher level of reuse. Although each individual technology has its suitable and well-defined theory or technique, what is the common and integrated language concept that supports the essence of this new development paradigm? How can we program directly with domain abstractions?

2. THE 4GL PARADIGM
When the domain maturates, the features are the communication and definition tool for understanding the common abstraction of the domain. The anatomy of a feature is a modular encapsulation of multi-dimensional views: an abstract view at the domain business level, a constructive view at the architectural pattern level and a concrete view at the implementation technologies level. Concrete features are implemented as software components. By observing that a language definition is a definition of the composition of language elements (tokens), we are motivated to use the language theory and techniques to define feature compositions (domain abstraction). Domain abstraction is analogous to a definition of a language; a particular feature composition instance is analogous to a program written in that language. We call these "domain abstraction languages" 4GLs.

1. The 4GLs have abstract and concrete forms. The abstract representation is the definition of domain abstraction defined by Two-Level Grammar++ (TLG++). A 4GL program's abstract form is encoded in XML. The concrete form of 4GLs can be one of the following: a model edited in a modeling language such as UML or GME [5], an online HTML form, a GUI wizard, a spreadsheet or simple text. Normally the abstract form can be generated from the concrete forms by the corresponding tool support by eliminating tool specifics. The abstract form of 4GL promotes reuse of domain abstractions across the tools. Historical 4GLs emphasized the concrete form and there was no uniform definition of 4GLs. Languages such as query languages, report generators, graphics languages, decision-support languages, application generators, application languages and specification languages were categorized as 4GLs [7]. The 4GLs were most popularized as data-query languages. As 4GLs varied drastically in form, there was no uniform means to describe the different syntax and semantics. Our work is focusing on the abstract form of 4GLs.

2. According to three-dimensional views of domain features, a 4GL program (a feature composition instance) has three-dimensional views too: semantic, syntactic, and lexical compositions referred to respectively as the 4GL semantics, syntax, and lexicality. The composition at each dimension is defined by that dimension's feature model. Composition at the lexical level mainly deals with the interoperation between the feature lexemes (software components).

3. RELATED WORK
GP provides the notion of a Generative Domain Model (GDM) that we employed as the result of domain engineering in 4Compiler, a 4GL compiler. However, GP dose not address how to organize the generated 3GL product into a functioning architectural organization while accommodating heterogeneous implementation technologies at the same time, where we claim our work will contribute. Batory has proposed feature oriented programming by static stepwise refinement on the base programs defined by refinement algebra [1], whereas, we are focusing on the dynamic feature composition defined by grammars. Composition Language (CL) [6] in Prediction Enabled

OOPSLA'03, October 26–30, 2003, Anaheim, California, USA.
ACM 1-58113-751-6/03/0010.

Component Technology described composition semantics in the component model level such as latency, safety and availability. Yet, CL did not address the composition semantics on the level of business meanings. DSLs offer language notations tailored towards the specific needs of a particular domain, but they are not defined directly by the composition of domain features. MDA focuses on the abstraction level of Meta (M0, M1, M2, M3) [4], whereas the basic theory of our work is emphasizing the abstraction level in engineering knowledge (business logic, architecture and technologies).

4. 4GL DEFINITION IN TLG++

The 4GL semantics and syntax are defined respectively by each dimension's feature models. We developed a new language called TLG++ as an object oriented extension to TLG [10] suitable for specifying feature models. The term "two-level" comes from the fact that a set of formal parameters may be defined using a context-free grammar, with the possible generated strings used as arguments in predicate functions defined using another context-free grammar. The second level, the rule of the first level for testing context sensitivity, has been extensively used in [10] to define the static semantics along with the language syntax. It has been proved and illustrated that TLG has Turing computation power that can be used as a grammatical interpretive model for dynamic semantics. The integration of syntax and semantics definition in a single grammatical notation is very convenient for specifying feature models. The composition syntax of the feature model is the domain feature organizational structure that is the functional perspective of the composition. The static semantics are configuration constraints such as feature attributes, relationship cardinalities, pre and post condition for the configurations, interdependencies and temporal concerns. The dynamic semantics of the composition models the stages of changes of system properties after the steps of composition, which has been called Quality-of-Service (QoS) composition [8]. Examples of QoS parameters are turn-around-time at the lexical composition level and the reliability at the syntactic composition level. The composition semantics is at the non-functional perspective of composition. Since both levels of TLG are context-free grammars, a TLG interpreter reads the feature model definition (grammatical interpretive model) and generates the 4GL semantics and syntax interpreter automatically by using parser generator facilities.

5. 4COMPILER—COMPILING THE 4GLS

A 4Compiler that reads a 4GL program (in a concrete form) and produces a 3GL object code is essentially a product–line assembler for that particular domain. 4Compiler has two phases. 1) The application development is a process of 4GL compilation. A 4GL program in a concrete form needs to be converted into the abstract form. It is first parsed according to semantic composition (no business logic violation), and secondly parsed according to syntactic composition (no architectural violation), and then transformed into an architecture representation with any necessary architectural instrumentation code generated automatically. Then the UniFrame Resource Discovery System [9] searches for the necessary feature implemented in the business domain search space. If there are any incompatibilities in the component models used in those feature implementations, the system will generate bridge code based on the knowledge from the technology GDM and parameters from the feature associated Unified Meta-component Model (UMM) [9]. 2) For 4GL compilation development, the domain level engineering phase simulates the domain development of three-dimensional domains (business domains, architecture domains and technology domains), which results in business GDM, architecture GDM and technology GDM, providing the 4GL semantic, syntactic and lexical definition feature model respectively. Concrete feature implementations are provided by designated programmers facilitated with MDA in business domains. Domain level development provides the meta-data and reusable assets for the application engineering.

6. CONCLUSIONS

Our prototyping starts from lexical composition. We have designed a framework so that the bridge code can be automatically and dynamically generated for interoperation between any pair of component models. The proposed Ph.D. research should be validated through a complete example that includes: a formal feature model definition in the Banking business domain; formal feature model definition in architectural pattern domain; and the ability to compile a sample 4GL program written in GME using the proposed methods. This research is supported by the U. S. Office of Naval Research under the award number N00014-01-1-0746.

7. REFERENCES

[1] D. Batory, J. N. Sarvela, A. Rauschmayer, "Scaling Step-Wise Refinement", Proc. of 25th International Conference on Software Engineering, pp.187-197, 2003.

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

[3] A. v. Deursen, P. Klint, J. Visser, "Domain-Specific Languages: An Annotated Bibliography", CWI, 2000, http://homepages.cwi.nl/~arie/papers/dslbib/#foot85

[4] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing.* Wiley Publishing, Inc., 2003.

[5] GME User's Manual. *The Institute for Software Integrated Systems, Vanderbilt University.* http://www.isis.vanderbilt.edu/Projects/gme/Doc.html

[6] J. Ivers, N. Sinha, K. Wallnau, "A Basis for Composition Language CL", Technical Note, CMU/SEI-2002-TN-026, 2002.

[7] James Martin, Fourth Generation Languages, Volume 1: Principles. Pretice-Hall, Inc., 1985.

[8] R. R. Raje, M. Auguston, B. R. Bryant, A. M. Olson, C. C. Burt, "A Quality of Service-Based Framework for Creating Distributed Heterogeneous Software Components," *Concurrency and Computation: Practice and Experience Vol. 14, No. 2,* pp. 1009-1034, 2002.

[9] UniFrame Project, http://www.cs.iupui.edu/uniFrame/

[10] A. van Wijngaarden, "Revised Report on the Algorithmic Language ALGOL 68." *Acta Informatica*, Vol. 5, pp. 1-236, 1974