# Natural Language Requirements to Executable Models of Software Components

V R Rathod S M Shah Nileshkumar K. Modi

**Abstract**

*The UniFrame approach to component-based software development assumes that concrete components are developed from a meta-model, called the Unified Meta- component Model, according to standardized business domain models. Implicit in this development is that there is a Platform Independent Model (PIM) which is transformed into a Platform Specific Model (PSM) under the principles of Model-Driven Architecture. This paper advocates natural language as the starting point for developing the business domain models and the meta-model and shows how this natural language may be mapped through the PIM to PSM using a formal system of rules expressed in Two-Level Grammar. This allows software requirements to be progressed from business logic to implementation of components and provides sufficient automation that components may be modified at the model level, or even the natural language requirements level, as opposed to the code level.*

**Keywords :** Natural Language Processing, Model-Driven Architecture

## 0. Introduction

Model-driven architecture (MDA) is an approach whereby software components are expressed using models, typically in UML. The basic approach is to define Platform Independent Models (PIMs) which express the business logic of components conforming to some domain (e.g. banking, telecommunications, etc.) and then to derive Platform Specific Models (PSMs) using a specific component technology (e.g. CORBA, J2EE, etc.). Business logic is typically expressed in natural language before a model is developed. Standardization of business domains and associated components is being undertaken by the Object Management Group (OMG). To facilitate the MDA approach to be used in practice, automated tools are needed to develop the business domain specifications from their requirements in natural language as well as to enable transformation from PIMs into PSMs. Furthermore, if MDA is to be used for constructing distributed software systems, then the models must consider not only functional aspects of business logic, but also non-functional aspects, which we call Quality-of-Service (QoS). QoS attributes are not currently considered in the MDA framework.

UniFrame is an approach for assembling heterogeneous distributed components, developed according to MDA principles, into a distributed software system with strict QoS requirements. Components are deployed on a network with an associated requirements specification, expressed as a Unified Meta-component Model (UMM) in the Two-Level Grammar (TLG) specification language. The UMM is integrated with generative domain models and generative rules for system assembly which may be automatically translated into an implementation which realizes an integration of components via generation of glue and wrapper code. Furthermore, the glue/wrapper code is instrumented to enable validation f the QoS requirements.

This paper describes a unified method of expressing business domain models in natural language, translating these into associated business logic rules for that domain, application of the business logic rules in building MDA PIMs, and maintaining these rules through development of PSMs. The complete mapping takes place using a formal system of rules expressed in Two-Level Grammar. This allows software requirements to be progressed from business logic to implementation of components and provides sufficient automation that components may be modified at the model level, or even the natural

language requirements level, as opposed to the code level. Section 2 describes our previous work with Two-Level Grammar and its use as a specification language. The application of this to Model-Driven Architecture is discussed in section 3. Finally we conclude in section 4.

## 1. From Natural Language Requirements to Formal Models

To achieve the conversion from requirements documents to formal models, several levels of conversions are required. First, the original requirements written in natural language are refined as a preprocessing of the actual conversion. This refinement task involves checking spellings, grammatical errors, consistent use of vocabularies, organizing the sentences into the appropriate sections, etc. The requirements are expected to be organized in a well-structured way, e.g. as laid out in or as a collection of use-cases, and be part of an ontological domain. Since we are allowing for specification of components that will be deployed in a distributed environment, Quality-of-Service attributes are also specified. Next the refined requirements document is expressed in XML format. By using XML to specify the requirements, XML attributes (meta-data) can be added to the requirements to interpret the role of each group of the sentences during the conversion. The information of the domain-specific knowledge is specified in XML. The domain-specific knowledge describes the relationship between components and other constraints that are presumed in requirements documents or too implicit to be extracted directly from the original documents. Then a knowledge base is built from the requirements document in XML using natural language processing (NLP) to parse the documentation and to store the syntax, semantics, and pragmatics information. In this phase, the ambiguity is detected and resolved, if possible. Once the knowledge base is constructed, its content can be queried in NL. Next, the knowledge base is converted with the information of the domain specific knowledge, into Two Level Grammar by removing the contextual dependency in the knowledge base. TLG is used as an intermediate representation to build a bridge between the informal knowledge base and the formal specification language representation. The name “two-level” in Two-Level Grammar comes from the fact that TLG consists of two context-free grammars interacting in a manner such that their combined computing power is equivalent to that of a Turing machine. Our work has refined this notion into a set of domain definitions and the set of function definitions operating on those domains. In order to support object-orientation, TLG domain declarations and associated functions may be structured into a class hierarchy supporting multiple inheritances.

Finally, the TLG code is translated into VDM++, an object-oriented extension of the Vienna Development Method, by data and function mappings. VDM++ is chosen as the target specification language because VDM++ has many similarities in structure to TLG and also has a good collection of tools for analysis and code generation. Once the VDM++ representation of the specification is acquired we can do prototyping of the specification using the VDM++ interpreter.

Also, we can convert this into a high level language such as Java or C++ or into a Rational Rose model in UML using the VDM++ Toolkit. Using XMI5 format, not only the class framework but also its detailed functionalities can be specified and translated into OCL (Object Constraint Language). The structure of the system is shown in Figure1.

## 2. Integration with Model-Driven Architecture

The method of translating requirements in natural language into UML models and/or executable code described in the previous section may be used to translate business logic into formal rules. Business domain experts from various application domains may express their specification in natural language and then our system translates this into Two-Level Grammar rules via natural language processing (NLP). These rules are encapsulated in a TLG class hierarchy defining a knowledge base with domain ontology, domain feature models (specifying the commonality and variability among the product instances in that domain), feature configuration constraints, feature interdependencies, business operational rules, temporal concerns, etc. TLG specifies the complete feature model including the structural syntax and various kinds of semantic concerns. For example, assume that our application domain is banking. The

business domain will then include a feature model of a bank, which includes specification of the various attributes and operations a bank will have, such as account creation and management, deposit, withdrawal and balance checking operations on individual accounts, etc.

In related work, we have investigated the construction of Generative Domain Models using the Generic Modeling Environment. This tool may also be extended with a natural language processor as a front end, i.e., by applying natural language processing to the business domain model (which is represented in natural language), which can then extract feature model representation rules and then interpret those rules to generate a graphical feature diagram. Platform Independent Models in MDA are based upon the business domains and associated logic for the given application. TLG allows these relationships to be expressed via inheritance. If a software engineer wants to design a server component to be used in bank account management systems, then he/she should write a natural language requirements specification in the form of a UMM (Unified Meta-component Model) describing the characteristics of that component. Our natural language requirements processing system will use the UMM and domain knowledge base to generate platform independent and platform specific UMM specifications expressed in TLG (which we will refer to as UMM-PI and UMM-PS, respectively). UMM-PI describes the bulk of the information needed to progress to component implementation. UMM-PS merely indicates the technology of choice (e.g. CORBA). These effectively customize the component model by inheriting from the TLG classes representing the business domain with new functionality added as desired.

In addition to new functionality, we also impose Quality-of-Service expectations for our components. Both the added functionality and QoS requirements are expressed in TLG so there is a unified notation for expressing all the needed information about components. The translation tool described in the previous section may be used to translate UMM-PI into a PIM represented by a combination of UML and TLG. Note that TLG is needed as an augmentation of UML to define business logic and other rules that may not be convenient to express in UML directly.

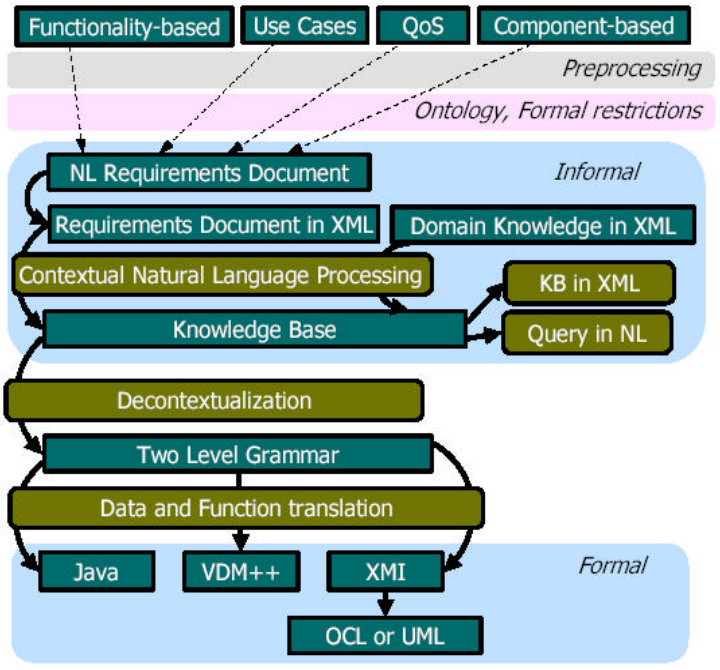


*Figure1. Natural Language Requirements Translation into Executable Models*

A Platform Specific Model is an integration of the PIM with technology domain specific operations (e.g. in CORBA, J2EE, etc.). These technology domain classes also are expressed in TLG. Each domain contains rules which are specific to that technology, including how to construct glue/wrapper code for components implemented with that technology and architectural considerations such as how to distinguish client code from server code. We express PSMs in TLG as an inheritance from PIM TLG classes and technology domain TLG classes. This means that PSMs will then contain not only the business-domain specific rules but also the technology domain specific rules. The PSM will also maintain the Quality-of-Service characteristics expressed at the PIM level (a related paper explores the rules for this maintenance in more detail and explores this issue for the QoS aspect of access control in particular).

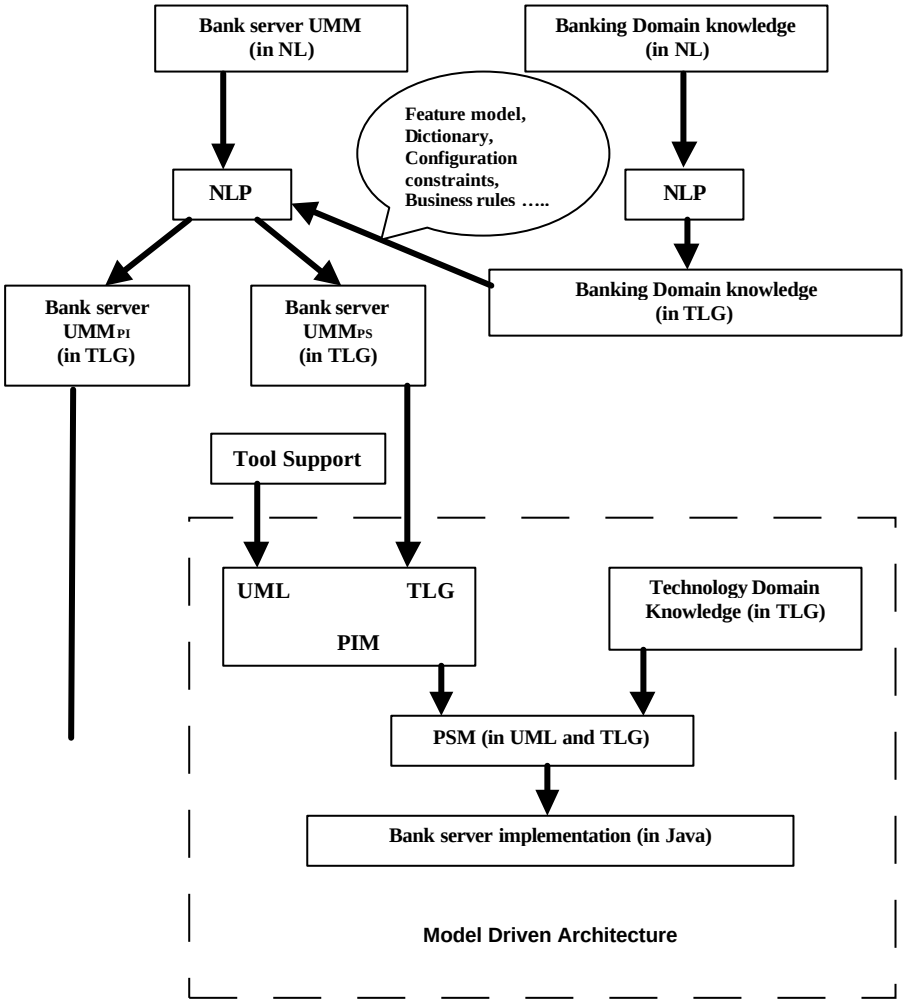


*Figure 2. Integration of Two-Level Grammar with Model Driven Architecture*

Since the model is expressed in TLG, it is executable in the sense that it may be translated into executable code in a high-level language (e.g. Java). Furthermore, it supports changes at the model level, or even requirements level if the model is not refined following its derivation from the requirements, since the code generation itself is automated.

Figure 2 shows the overall view of the model-driven development from natural language requirements into executable code for the banking example we have just described.

## 3. Conclusion

This paper has described an approach for unifying the ideas of expressing requirements in natural language, constructing Platform Independent Models for software components, and implementing the components via Platform Specific Models. The approach is specifically targeted at the construction of heterogeneous distributed software systems where interoperability is critical. This interoperability is achieved by the formalization of technology domains with rules describing how those technologies may be integrated together via the generation of glue and wrapper code. The processing of software requirements, construction of PIMs and PSMs, and specification of technology domain rules are all expressed in Two-Level Grammar, thereby achieving a unification of natural language requirements with the Model Driven Architecture approach.

For future work, we will investigate aspect-oriented technology as a mechanism or specifying crosscutting relationships across components and hence improving reusability of components and reasoning about a collection of components. Such aspects of components as functional pre/post conditions and QoS properties crosscut component modules and specification of these aspects spread across component modules. Preliminary work in defining an aspect oriented specification language is very promising.

We are also investigating the applicability of the UniFrame approach to real-time and embedded systems. Real-time constraints are already one of the Quality-of-Service parameters we are now validating. However, we expect that our current timing requirements will need refinement to be applicable in a true real-time setting. We are also looking at applying our modeling technology to the embedded system domain. Finally we are continuing our work in model-driven security to assure that security issues are maintained in migration from PIMs to PSMs.

## 4. References

1. “Two-Level Grammar as an Object-Oriented Requirements Specification Language,” Proc. HICSS-35, 35th Hawaii Int. Conf. System Sciences, 2002, http://www.hicss.hawaii.edu/HICSS_35/HICSSpapers/PDFdocuments/STDSL01.pdf.
2. “Formal Specification of Generative Component Assembly Using Two-Level Grammar,” Proc. SEKE 2002, 14th Int. Conf. Software Engineering Knowledge Engineering, 2002, pp. 209-212.
3. Burt, C. C., Bryant, B. R., Raje, R. R., Olson, A. M., Auguston, M., “Model Driven Security: Unification of Authorization Models for Fine-Grain Access Control,” to appear in Proc. EDOC 2003, 7th IEEE Int. Enterprise Distributed Object Computing Conf.
4. “Automating Feature-Oriented Domain Analysis ,” to appear in Proc. SERP 2003, 2003 Int. Conf. Software Engineering Research and Practice, 2003.
5. “Assembling Components with Aspect-Oriented Modeling/Specification,” to appear in Proc. WiSME 2003, UML 2003 Workshop Software Model Engineering.
6. “VDM++ - A Formal Specification Language for Object-Oriented Designs,” Proc. TOOLS USA ’92, 1992 Technology of Object-Oriented Languages and Systems USA Conf., 1992, pp. 263-278.
7. GME 2000 User’s Manual, Version 2.0. ISIS, Vanderbilt University, 2001.
8. IFAD, The VDM++ Toolbox User Manual, http://www.ifad.dk, 2000.

9. Jacobson, I., Booch, G., Rumbaugh, J., The Unified Software Development Process, Addison-Wesley, 1999.

10. Jurafsky, D., Martin, J., Speech and Language Processing, Prentice-Hall, 2000.

11. Kiczales, G., et al., “Aspect-Oriented Programming,” Proc. ECOOP ’97, European Conf. Object-Oriented Programming, 1997, pp. 220-242.

12. Lee, B.-S. and Bryant, B. R., “Contextual Knowledge Representation for Requirements

13. Documents in Natural Language,” Proc. FLAIRS 2002, 15th Int. Florida AI Research Symp., 2002, pp. 370-374.

14. Lee, B.-S. and Bryant, B. R., “Contextual Processing and DAML for Understanding Software Requirements Specifications,” Proc. COLING 2002, 19th Int. Conf. Computational Linguistics, 2002, pp. 516-522.

15. Lee, B.-S., Bryant, B. R., “Automation of Software System Development Using Natural Language Processing and Two-Level Grammar,” Proc. 2002 Monterey Workshop Radical Innovations Software and Systems Engineering in the Future, 2002, pp. 244-257.

16. Quatrani, T., Visual Modeling with Rational Rose 2000 and UML, Addison-Wesley, Reading, MA, 2000.

17. Raje, R. R., “UMM: Unified Meta-object Model for Open Distributed Systems,” Proc. ICA3PP, 4th IEEE Int. Conf. Algorithms and Architecture for Parallel Processing, 2000, pp. 454- 465.

18. Raje, R. R., Auguston, M., Bryant, B. R., Olson, A. M., and Burt, C. C., “A Unified Approach for the Integration of Distributed Heterogeneous Software Components,” Proc. 2001

19. Monterey Workshop Engineering Automation for Software Intensive System Integration, 2001, pp. 109-119.

20. Raje, R. R., Auguston, M., Bryant, B. R., Olson, A. M., Burt, C. C., “A Quality of Service-based Framework for Creating Distributed Heterogeneous Software Components,”Concurrency and Computation: Practice and Experience 14, 12 (2002), 1009-1034.

21. Warmer, J., Kleppe, A., The Object Constraint Language: Precise Modeling with UML, Addison-Wesley, 1999.

22. Wilson, W. M., “Writing Effective Natural Language Requirements Specifications,” Naval Research Laboratory, 1999.

23. Yang, C., Lee, B.-S., Bryant, B. R., Burt, C. C., Raje, R. R., Olson, A. M., Auguston, M., “Formal Specification of Non-Functional Aspects in Two-Level Grammar,” Proc.

24. UML 2002 Workshop Component-Based Software Engineering and Modeling Non-FunctionalAspects(SIVOES-MONA),2002,http://www-verimag.imag.fr/SIVOES-ONA/uniframe.pdf.

25. UML – Unified Modeling Language, http://www.omg.org/uml

27 CORBA – Common Object Request Broker Architecture, http://www.corba.org

28. J2EE – Java 2 Enterprise Edition, http://java.sun.com/j2ee

## About Authors

**Dr. V R Rathod** is a Professor and Head in Department of Computer Science, Bhavnagar University, Bhavnagar, Gujarat.
**E-mail :** profvrr@rediffmail.com

**Prof. S M Shah** is working as Director in S. V. Institute of Computer Studies, S. V. Campus, Kadi. Gujarat.
**E-mail :** prof_smshah@yahoo.com

**Mr. Nileshkumar K Modi** is a Lecturer in S. V. Institute of Computer Studies, S. V. Campus, Kadi, Gujarat.
**E-mail :** tonileshmodi@yahoo.com