From Natural Language Requirements to Executable Models of Software Components

Barrett R. Bryant, Beum-Seuk Lee, Fei Cao, Wei Zhao, Jeffrey G. Gray, Carol C. Burt
University of Alabama at Birmingham
{bryant, leebs, caof, zhaow, gray, cburt}@cis.uab.edu

Rajeev R. Raje, Andrew M. Olson
Indiana University-Purdue University-Indianapolis
{rraje, aolson}@cs.iupui.edu

Mikhail Auguston
Naval Postgraduate School
auguston@cs.nps.navy.mil

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 domain models. Implicit in this development is the existence of a Platform Independent Model (PIM) that is transformed into a Platform Specific Model (PSM) under the principles of Model-Driven Architecture (MDA). This position paper advocates natural language as the starting point for developing the meta-model and representative domain models. The paper illustrates how natural language is mapped through the PIM to PSM using a formal system of rules expressed in a Two-Level Grammar (TLG). This allows software requirements to be progressed from domain logic to the implementation of components. The approach provides sufficient automation such that components may be modified at the model level, or even the natural language requirements level, as opposed to the code level.

1. Introduction

Model-Driven Architecture (MDA) [12] is an approach that separates the essence of an application from the specific middleware platform to which it is deployed. The basic approach is to define Platform Independent Models (PIMs) that express the application logic of components conforming to some domain (e.g., mission-computing avionics, safety-critical medical devices) and then to derive Platform Specific Models (PSMs) using a specific component technology (e.g. CORBA [1], J2EE [2], and .NET [3]).

Domain logic is typically expressed in natural language before a model is developed. Standardization of domains and their associated components is being undertaken by the Object Management Group (OMG) [4]. To facilitate the MDA approach to be used in practice, automated tools are needed to develop the 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 real-time embedded (DRE) software systems, then the models must consider not only functional aspects of domain logic, but also non-functional properties, such as Quality-of-Service (QoS) requirements (e.g., latency and bandwidth requirements on a distributed video streaming system [23]). QoS attributes are not currently considered in the MDA framework.

UniFrame [31] 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) [30] in the Two-Level Grammar (TLG) specification language [4]. The UMM is integrated with generative domain models and generative rules for system assembly [10], which may be automatically translated into an implementation that realizes an integration of components via generation of glue and wrapper code. Furthermore, the glue/wrapper code is instrumented to enable validation of the QoS requirements [32].

This paper describes a unified method of expressing domain models in natural language, translating these into associated logic rules for that domain, application

[1] CORBA – Common Object Request Broker Architecture, http://www.corba.org

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

[3] http://www.microsoft.com/net

[4] http://www.omg.org

of the 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 TLG. This allows software requirements to be progressed from domain logic to implementation of components. It also provides sufficient automation such 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 TLG and its use as a specification language. The application of this to MDA is discussed in section 3. Finally, we conclude in section 4.

2. From Natural Language Requirements to Formal Models

To achieve the conversion from requirements documents to formal models requires several levels of conversion, as shown in Figure 1. 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, and organizing the sentences into the appropriate sections. The requirements are expected to be organized in a well-structured way, e.g. as laid out in [36] or as a collection of use-cases [16], and be part of an ontological domain [21]. Once they are structured in this way via human preprocessing, the remainder of the conversion occurs automatically. If modifications to requirements are needed, these modifications should be made to the requirements already preprocessed, not the original ones. Since we are allowing for specification of components that will be deployed in a distributed environment, Quality-of-Service attributes are also specified [38].

An example requirements specification from [19] is given below. This is a small piece of the Computer Assisted Resuscitation Algorithm (CARA) Infusion Pump Control System [37].

```
The host is powered up and all
software subsystems are available.
The pump software system is now in
the wait operating state. The patient
with IV/pump running is placed onto
the host. The pump cable is connected
to the host. The host now provides
power for the pump.
```

Next, the refined requirements document is automatically converted into XML[5] 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 to exist in requirements documents or too implicit to be extracted directly from the original documents [22]. The XML representation produced for the above specification is:

```
<class title = "Mode" meta = "mode">
  <class title = "wait state" meta
      = "submode">
    <paragraph meta = "pre_cond">
      <sentence>
        Host is powered up and all
        software subsystems are
        available
      </sentence>
    </paragraph>
    <paragraph meta = "pre_exec">
      <sentence>
        Patient with IV/pump
        running is placed onto the
        host
      </sentence>
      <sentence>
        Pump cable is connected to
        the host
    </sentence>
    </paragraph>
    <paragraph meta = "exec">
      <sentence>
        HOST now provides power for
        pump
      </sentence>
    </paragraph>
  </class>
  …
</class>
```

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. Each sentence is read by the system and each sentence is parsed into words. At the syntactical level, the part of speech (e.g. noun, verb, adjective) of each word is determined by bottom-up parsing, whereas the part of sentence (e.g. subject, object, complement) of each word is determined by top-down parsing [17]. The corpora of statistically ordered parts of speech (frequently used ones being listed first) of about 85,000

[5] XML – eXtensible Markup Language – http://www.w3c.org/xml

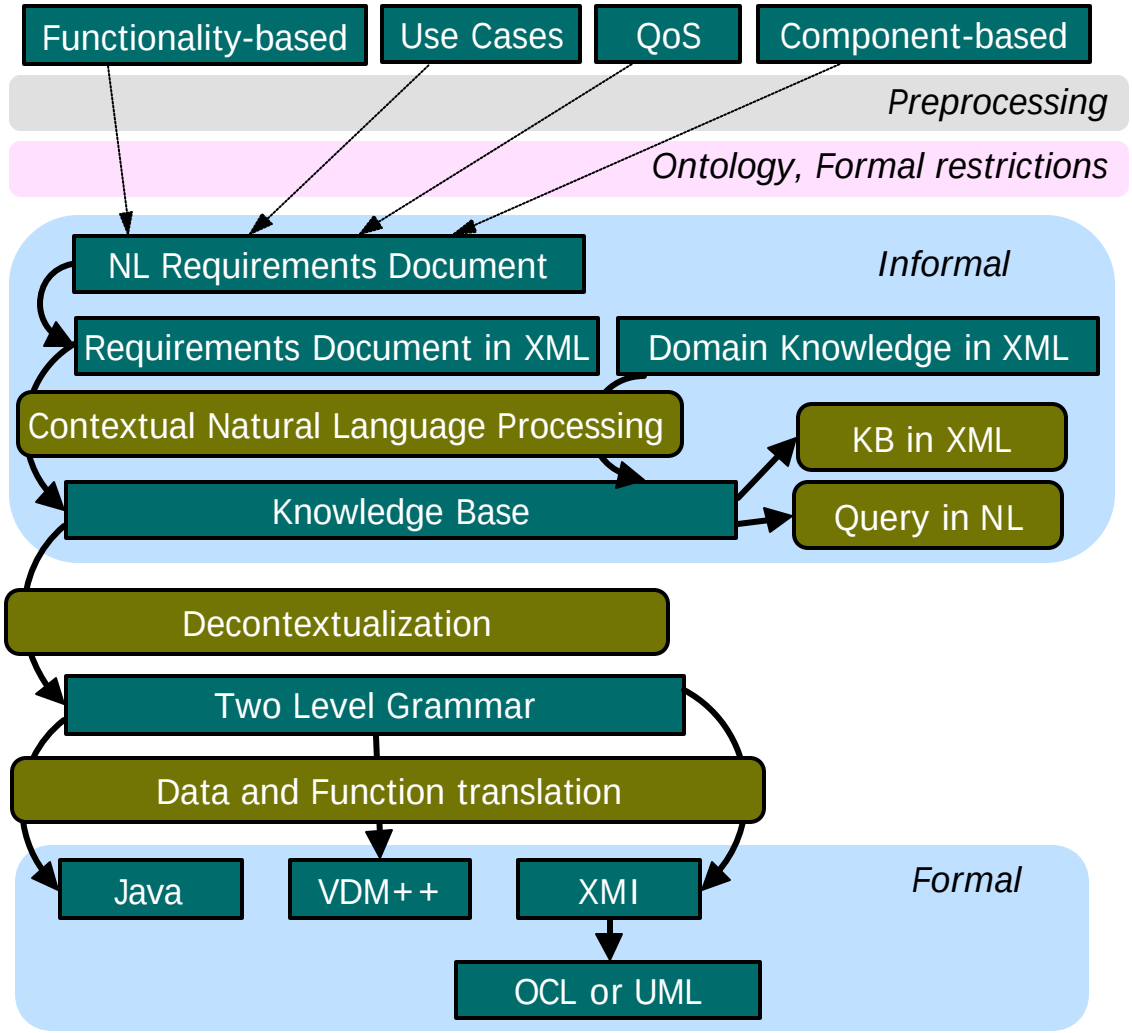

Figure 1. Natural Language Requirements Translation into Executable Models

words from [34] are used to resolve syntactic ambiguities in this phase. Also, elliptical compound phrases, comparative phrases, compound nouns, and relative phrases are handled in this phase as well. The knowledge base for the above example is shown in Figure 2.

Once the knowledge base is constructed, its content can be queried in NL. Next, the knowledge base is converted, with the domain-specific knowledge, into TLG by removing contextual dependencies in the knowledge base [20]. 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 TLG 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

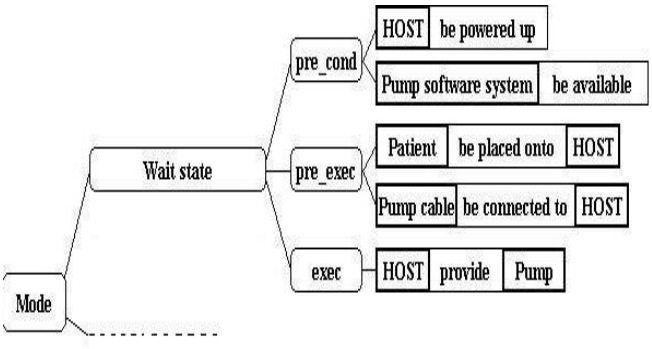

Figure 2: Knowledge Representation

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 inheritance. The TLG specification produced for this example is:

```
class Mode.
  wait state :
    Host is powered up,
    Pump SoftwareSystem is
      available,
    Patient with IVPump running
      is placed onto Host,
    Pump Cable is connected to Host,
      Host provides Power for Pump.
    …
end class Mode.
```

Host, Pump, SoftwareSystem (an attribute of Pump), Patient, IVPump (an attribute of Patient), Cable (an attribute of Pump), and Power have all been identified as objects in the analysis. In TLG, object and class names are denoted by being capitalized (and are in fact not distinguished, i.e., an object may be denoted using the corresponding class name, as an implicit declaration). Verbs and other words are included in TLG to make up functions, e.g. "is powered up," "is available," etc.

As a final step in this process, the TLG code is translated into VDM++, an object-oriented extension of the Vienna Development Method [11], 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, prototyping can be performed on the specification using the VDM++ interpreter to validate the generated formal specification against the original requirements. Also, the formal VDM++ representation can be converted into a high level language such as Java or C++, or into a Rational Rose model in UML[6] [29] using the VDM++ Toolkit [15]. The VDM++ specification created for the above TLG is:

```
class Mode

instance variables
  private host : Host
  private pump : Pump
  private patient : Patient
  private power : Power

operations
  …
  public waitState : () => ()
  waitState () == (
    host . poweredUp ();
    pump . softwareSystem ()
      . available ();
    patient . ivPump ()
      . running ();
    patient . placedOnto (host);
    pump . cable ()
      . connectedTo (host);
    host . provides (power, pump);
  );
  …

end class Mode
```

The VDM++ class uses one instance variable to represent each object in the TLG specification. This VDM++ specification may be converted into the UML model shown in Figure 3. Using the XMI[7] format, not only the class framework but also its detailed functionalities can be specified and translated into OCL (Object Constraint Language) [35].

3. Integration with Model-Driven Architecture

The method of translating requirements in natural language into UML models and/or executable code (as described in the previous section) may be used to translate domain logic into formal rules. Experts from various application domains may express their specification in natural language and then use UniFrame to translate this into TLG rules via natural language processing. These rules are encapsulated in a TLG class hierarchy defining a knowledge base with the domain ontology, domain feature models (specifying the commonality and variability among the product instances in that domain), feature configuration constraints, feature interdependencies, operational rules, and temporal

[6] UML – Unified Modeling Language, http://www.omg.org/uml

[7] XMI - XML Metadata Interchange, http://www.omg.org/technology/documents/formal/xmi.htm

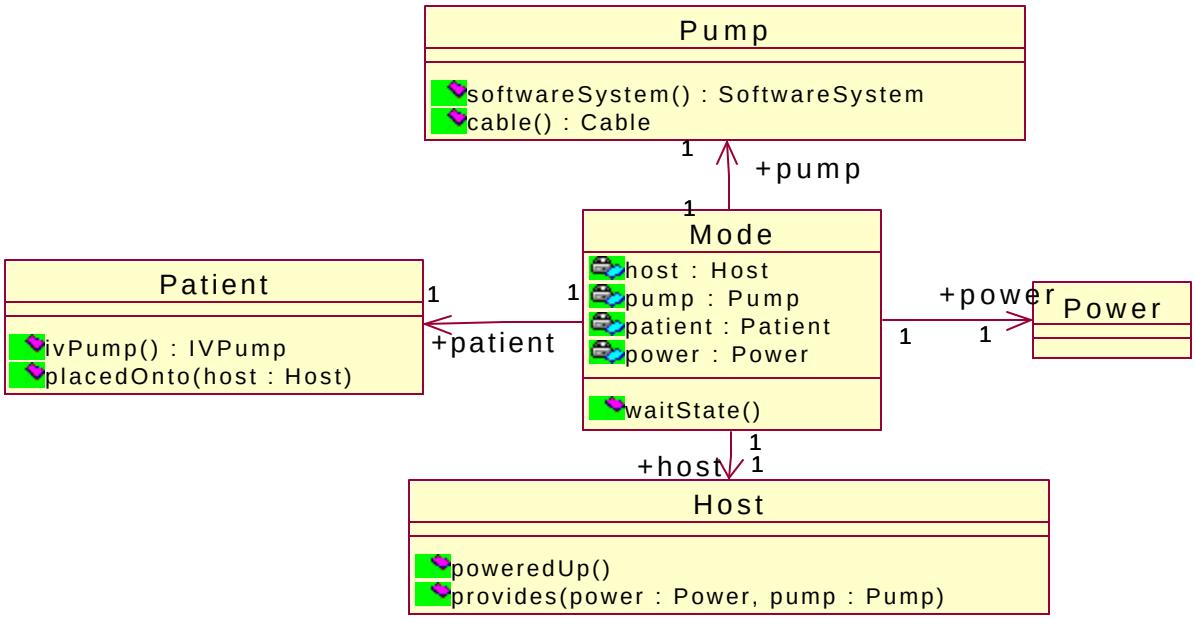

Figure 3: UML Representation of Requirements

concerns. TLG specifies the complete feature model including the structural syntax and various kinds of semantic concerns [39]. For example, assume that our application domain is for unmanned aerial vehicles (UAV's). The business domain will then include a feature model of a UAV, which includes specification of the various attributes and operations a UAV will have, such as responding to external commands and streaming video back to a satellite receiver [23]. In related work [8], we have investigated the construction of Generative Domain Models [10] using the Generic Modeling Environment [14]. This tool may also be extended with a natural language processor as a front end, i.e., by applying natural language processing to the domain model (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 (PIM's) in MDA are based upon the 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 a distributed video streaming application, 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 domain with new functionality added as desired. In addition to new functionality, we also impose end-to-end Quality-of-Service expectations for our components (e.g., a specification of the minimum frame-rate in a distributed video streaming application). 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 domain logic and other rules that may not be convenient to express in UML directly.

A Platform Specific Model (PSM) is an integration of the PIM with technology domain-specific operations (e.g. in CORBA, J2EE, or .NET). These technology domain classes also are expressed in TLG. Each domain contains rules that are specific to that technology, including how to construct glue/wrapper code for components implemented with that technology. Architectural considerations are also specified, 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 application-domain-specific rules, but also the technology-domain-specific rules. The PSM will also maintain the QoS characteristics

expressed at the PIM level (a related paper [6] explores the rules for this maintenance in more detail and [7] explores this issue for the QoS aspect of access control in particular). Because 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, because the code generation itself is automated.

Figure 4 shows the overall view of the model-driven development from natural language requirements into executable code for the previously described distributed video streaming application.

4. Related Work and Discussion

The idea of using natural language as the basis for developing software dates back at least 20 years. Abbott [1] pointed out that nouns correspond to the

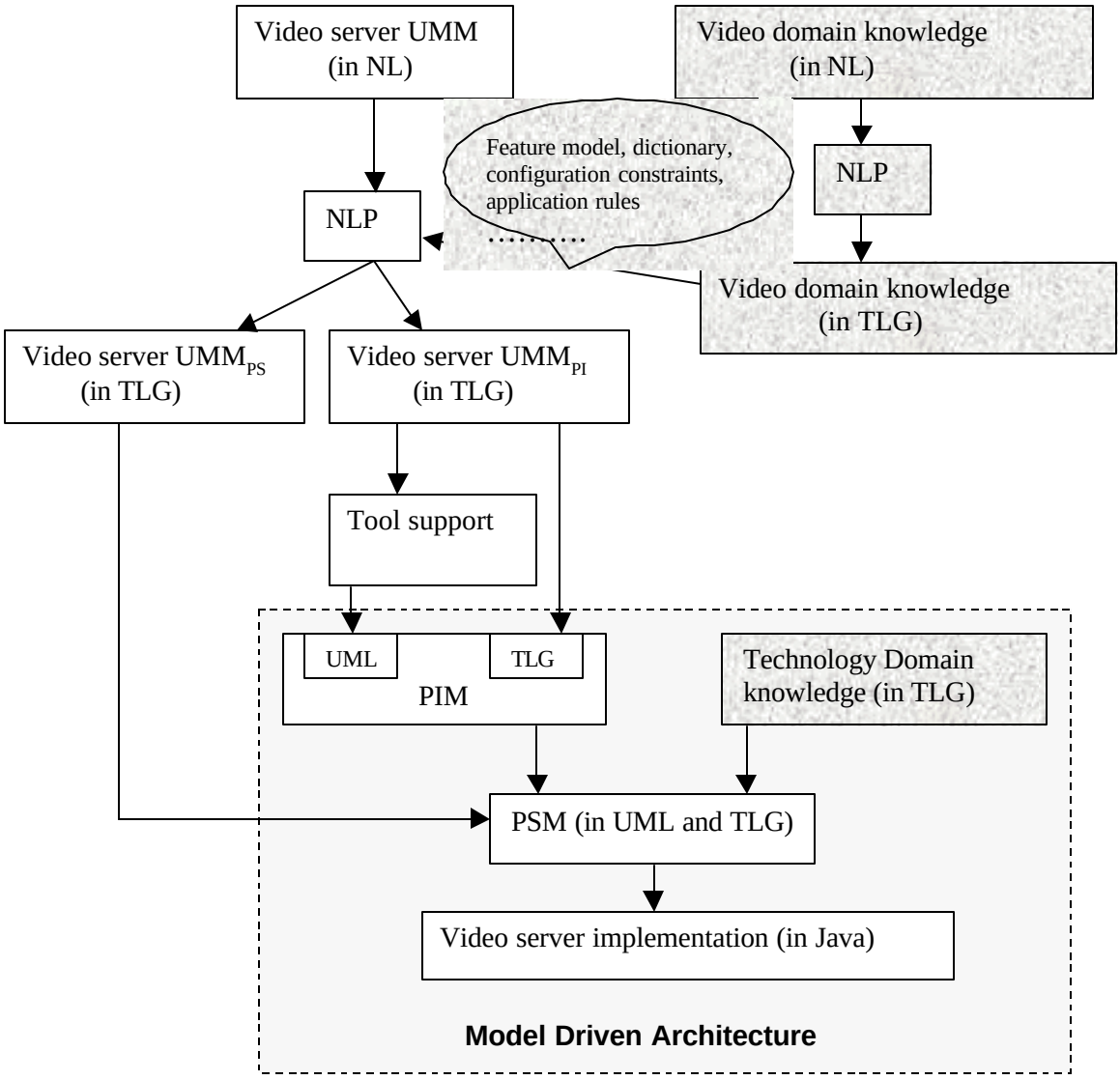

Figure 4. Integration of Two-Level Grammar with Model Driven Architecture

notion of a class in object-oriented programming terminology, direct references correspond to objects, while verb and attributes correspond to class operations, and the control flow within those operations is also often present in the action description. Rolland and Proix [33] developed an automated tool called OICSI[8], which facilitated the elicitation of requirements from natural language text and accompanying domain knowledge. Luisa Mich and her colleagues ([24], [25], [26]) have used a natural language processing system called NL-OOPS to analyze natural language requirements for the purpose of determining objects and their inter-relationships and construction of a corresponding object-oriented model. Nanduri and Rugaber [27] implemented a similar system for the purpose of validating an object-oriented model against the natural language requirements from which it was derived. Ambriola and Gervasi [2] extended this idea to incorporate modeling and model checking to achieve a more formal validation (the authors use the term "semi-formal" to describe the validation approach, which eventually evolved into "lightweight formal methods" [13]). LIDA (Linguistic Assistant for Domain Analysis) [28] appears to be the most comprehensive system to date for assisting a software engineer to construct an object-oriented model from natural language descriptions, the emphasis being on domain models. Daniel Berry and his colleagues (e.g., see [3]) have also worked with the problem of analyzing natural language specifications and have identified a number of difficult problems in correctly implementing requirements based upon natural language.

Our work has focused on conversion of natural language to formal specifications in VDM++, which in turn may be converted into UML models or executable code. 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 TLG, thereby achieving a unification of natural language requirements with MDA.

For future work, we will investigate aspect-oriented technology [18] as a mechanism for 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 [9].

[8] French acronym for intelligent tool for information system design," also called ALECSI [33]

5. Acknowledgements

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 number DAAD19-00-1-0350, and by the U. S. Office of Naval Research under award number N00014-01-1-0746.

6. References

[1] Abbott, R. J., "Program Design by Informal English Descriptions," *Commun. ACM 26*, 11 (Nov. 1983), 882-894.

[2] Ambriola, V. and Gervasi, V., "Processing Natural Language Requirements," *Proc. ASE '97, 12th Int. Conf. Automated Software Engineering*, 1997, pp. 36-45.

[3] Berry, D. M. and Kamsties, E., "Ambiguity in Requirements Specification," *Perspectives on Software Requirements*, eds. J. C. Sampaio do Prado Leite and J. H. Doorn, Kluwer Academic, 2003, pp. 191-194.

[4] Bryant, B. R. and Lee, B.-S., "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.

[5] Bryant, B. R., Auguston, M., Raje, R. R., Burt, C. C., and Olson, A. M., "Formal Specification of Generative Component Assembly Using Two-Level Grammar," *Proc. SEKE 2002, 14th Int. Conf. Software Engineering Knowledge Engineering*, 2002, pp. 209-212.

[6] Burt, C. C., Bryant, B. R., Raje, R. R., Olson, A. M., Auguston, M., "Quality of Service Issues Related to Transforming Platform Independent Models to Platform Specific Models," *Proc. EDOC 2002, 6th IEEE Int. Enterprise Distributed Object Computing Conf.*, 2002, pp. 212-223.

[7] 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," *Proc. EDOC 2003, 7th IEEE Int. Enterprise Distributed Object Computing Conf.*, 2003, pp. 159-171.

[8] Cao, F., Bryant, B. R., Burt, C. C., Huang, Z., Raje, R. R., Olson, A. M., Auguston, M., "Automating Feature-Oriented Domain Analysis," *Proc. SERP 2003, 2003 Int. Conf. Software Engineering Research and Practice*, 2003 , pp. 944-949.

[9] Cao, F., Bryant, B. R., Raje, R. R., Auguston, M., Olson, A. M., Burt, C. C., "Assembling Components with Aspect-Oriented Modeling/Specification," *Proc. WiSME 2003, UML 2003 Workshop Software Model Engineering*, 2003, http://www.metamodel.com/wisme-2003/12.pdf.

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

[11] Dürr, E. H., van Katwijk, J., "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.

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

[13] Gervasi, V. and Nuseibeh, B., "Lightweight Validation of Natural Language Requirements," *Softw. Pract. Exper. 32* (2002), 113-133.

[14] *GME 2000 User's Manual, Version 2.0*. ISIS, Vanderbilt University, 2001, http://www.isis.vanderbilt.edu/publications/archive/Ledeczi_A_12_18_2001_GME_2000_U.pdf.

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

[16] Jacobson, I., Booch, G., Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, 1999.

[17] Jurafsky, D., Martin, J., *Speech and Language Processing*, Prentice-Hall, 2000.

[18] Kiczales, G., et al., "Aspect-Oriented Programming," *Proc. ECOOP '97, 1997 European Conf. Object-Oriented Programming*, 1997, pp. 220-242.

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

[20] Lee, B.-S. and Bryant, B. R., "Contextual Knowledge Representation for Requirements Documents in Natural Language," *Proc. FLAIRS 2002, 15th Int. Florida AI Research Symp.*, 2002, pp. 370-374.

[21] 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.

[22] Lee, B.-S. and Bryant, B. R., "Applying XML Technology for Implementation of Natural Language Specifications," *Comput. Syst., Sci. & Eng. 5* (September 2003), 3-24.

[23] Loyall, J., Schantz, R., Atighetchi, M., and Pal, P., "Packaging Quality of Service Control Behaviors for Reuse," *Proc. ISORC 2002, 5th IEEE Int. Symp.Object-Oriented Real-time Distributed Computing*, 2002, pp. 375-385.

[24] Mich, L., "NL-OOPS: From Natural Language to Object-Oriented Requirements using the Natural Language Processing System LOLITA," *J. Nat. Lang. Eng. 2*, 2 (1996), 161-187.

[25] Mich, L. and Garigliano, R., "The NL-OOPS Project: OO Modeling using the NLPS LOLITA," *Proc. NLDB '99, 4th Int. Conf. Applications of Natural Language to Information Systems*, 1999, pp. 215-218.

[26] Mich, L., Mylopoulos, J., and Zeni, N., "Improving the Quality of Conceptual Models with NLP Tools: An Experiment," Technical Report, Department of Information and Communication Technologies, University of Trento, Italy, 2002, http://eprints.biblio.unitn.it/archive/00000127/01/ 47.pdf.

[27] Nanduri, S. and Rugaber, S., "Requirements Validation via Automated Natural Language Parsing," *J. Manage. Inf. Syst. 12*, 2 (1996), 9-19.

[28] Overmyer, S. P., Lavoie, B., and Rambow, O., "Conceptual Modeling through Linguistic Analysis using LIDA," *Proc. ICSE 2001, 23rd Int. Conf. Software Engineering*, 2001, pp. 401-410.

[29] Quatrani, T., *Visual Modeling with Rational Rose 2000 and UML*, Addison-Wesley, Reading, MA, 2000.

[30] 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.

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

[32] 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 Comput.: Pract. Exp. 14*, 12 (2002), 1009-1034.

[33] Rolland, C. and Proix, C., "A Natural Language Approach for Requirements Engineering," *Proc CAiSE '92, 4th Int. Conf. Advanced Information Systems*, 1992.

[34] Ward, G., "Moby Part-of-Speech II (data file)," 1994, http://www.gutenberg.net/extext02/mposp10.zip.

[35] Warmer, J., Kleppe, A., *The Object Constraint Language: Precise Modeling with UML*, Addison-Wesley, 1999.

[36] Wilson, W. M., "Writing Effective Natural Language Requirements Specifications," Naval Research Laboratory, 1999.

[37] Walter Reed Army Institute for Research (WRAIR), "CARA Specification: Proprietary Document," WRAIR, Dept. of Resuscitative Medicine, 2001.

[38] 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. UML 2002 Workshop Component-Based Software Engineering and Modeling Non-Functional Aspects (SIVOES-MONA)*, 2002, http://www-verimag.imag.fr/SIVOES-MONA/uniframe.pdf.

[39] Zhao, W., Bryant, B. R., Burt, C. C., Gray, J. G., Raje, R. R., Olson, A. M., Auguston, M. "A Generative and Model Driven Framework for Automated Software Product Generation," *Proc. CBSE 6, 6th Workshop Component-Based Software Engineering*, 2003, http://www.csse.monash.edu.au/~hws/cgi-bin/CBSE6/Proceedings/papersfinal/p31.pdf.