92-21

A Knowledge-Based Approach to Generating Target System Specifications from a Domain Model

Hassan Gomaa, Larry Kerschberg, and Vijayan Sugumaran

Center for Software Systems Engineering
Department of Information and Software Systems Engineering
George Mason University, Fairfax, Virginia 22030-4444, USA

Abstract

Several institutions in industry and academia are pursuing research efforts in domain modeling to address unresolved issues in software reuse. To demonstrate the concepts of domain modeling and software reuse, a prototype software engineering environment is being developed at George Mason University to support the creation of domain models and the generation of target system specifications. This prototype environment, which is application domain independent, consists of an integrated set of commercial off-the-shelf software tools and custom developed software tools. This paper describes the knowledge-based tool that has been developed as part of the environment to generate target system specifications from a domain model.

Keyword Codes: D.2.1; D.2.2; I.2.1
Keywords: Requirements/Specifications; Tools and Techniques; Applications and Expert Systems.

1. INTRODUCTION

An application domain is defined to be a collection of systems that share common features. A domain model is used to capture common characteristics and variations among a family of software systems in a given application domain. From the domain model, a target system can be generated by tailoring the domain model given the requirements of the target system. Thus, a target system engineer can develop the specification for a target system in terms of the domain model specified previously by a domain analyst, and does not have to perform a full systems analysis every time a new target system has to be constructed.

At George Mason University, a project is underway to support software engineering life cycles, methods, and prototyping environments to support software reuse at the requirements and design phases of the software lifecycle, in addition to the coding phase. A reuse-oriented software lifecycle, the Evolutionary Domain Lifecycle (Gomaa, 1989; Gomaa, 1991a) has been proposed, which is a highly iterative lifecycle that takes an application domain perspective allowing the development of families of systems. This paper describes a knowledge-based approach for generating target system specifications from a domain model.

Accepted for publication at the IFIP Congress, Madrid Spain, 1992

2. DOMAIN ANALYSIS AND MODELING

The Evolutionary Domain Life Cycle (EDLC) Model (Gomaa, 1989) is a software life cycle model that eliminates the traditional distinction between software development and maintenance. Instead, systems evolve through several iterations. Hence, systems developed using this approach need to be capable of adapting to changes in requirements during each iteration. Furthermore, because new software systems are often outgrowths of existing ones, the EDLC model takes an application domain perspective allowing the development of families of systems.

Parnas referred to a collection of systems that share common characteristics as a family of systems (Parnas, 1979). According to Parnas, it is worth considering a family of systems when there is more to be gained by analyzing the systems collectively rather than separately, i.e. the systems have more features in common than features that distinguish them. The concept of viewing an application domain as consisting of a family of systems has been adopted by various researchers (Batory, 1989; Pyster, 1990; Lubars, 1989).

When considering the development of a family of systems, it is necessary to replace the traditional system development activities of Requirements Analysis, Requirements Specification, System Design by activities that span the whole application domain. These are Domain Analysis, Domain Specification and Domain Design.

A Domain Model is a problem-oriented architecture for the application domain that reflects the similarities and variations of the members of the domain (Gomaa 1992). Given a domain model of an application domain, an individual target system (one of the members of the family) is created by tailoring the domain model given the requirements of the individual system. A Domain Model is initially created by means of a Domain Analysis. Domain Analysis (Prieto-Diaz, 1987) is a requirements analysis that addresses the requirements of a family of systems that are likely to be required for the domain, rather than a given target system.

2.1. Multiple Views of Domain Model

Applying the domain modeling method, the application domain is modeled by means of the following views (Gomaa 1992):

- *Aggregation Hierarchy.* The Aggregation Hierarchy is used to decompose complex aggregate object types into less complex object types eventually leading to simple object types at the leaves of the hierarchy.
- *Object Communication Diagrams.* Objects in the real world are modeled as concurrent processes, which communicate with each other using messages. The object communication diagrams, which are hierarchically structured, show how objects communicate with each other.
- *State Transition Diagrams.* As each active object is modeled as a sequential process, it may be defined by means of a state transition diagram, whose execution is strictly sequential.
- *Generalization / Specialization Hierarchies.* As the requirements of a given object type are changed to meet the needs of a given target system, the object type may be specialized by adding, modifying or suppressing operations. The variants of a domain object type are stored in this hierarchy.
- *Feature / Object Dependencies.* This view shows for each feature (domain requirement) the object types required to support the feature.

The domain modeling method has been applied to developing a domain model for NASA's Payload Operations Control Center (POCC) Domain.

3. PROTOTYPE SOFTWARE ENGINEERING ENVIRONMENT

A prototype software engineering environment is being developed, which consists of an integrated set of software tools that support domain modeling and the generation of target system specifications. A schematic representation of the prototype environment is given in Figure 1. In order to expedite development of the prototype, the environment uses commercial-of-the-shelf software as well as custom developed software. We are using Interactive Development Environments' Software Through Pictures (StP) CASE tool to represent the multiple views of the domain model, although semantically interpreting the views according to the domain modeling method. The information in the multiple views is extracted, checked for consistency, and mapped to an underlying representation, referred to as the domain specification, which is stored in an object repository (Gomaa, 1991b).

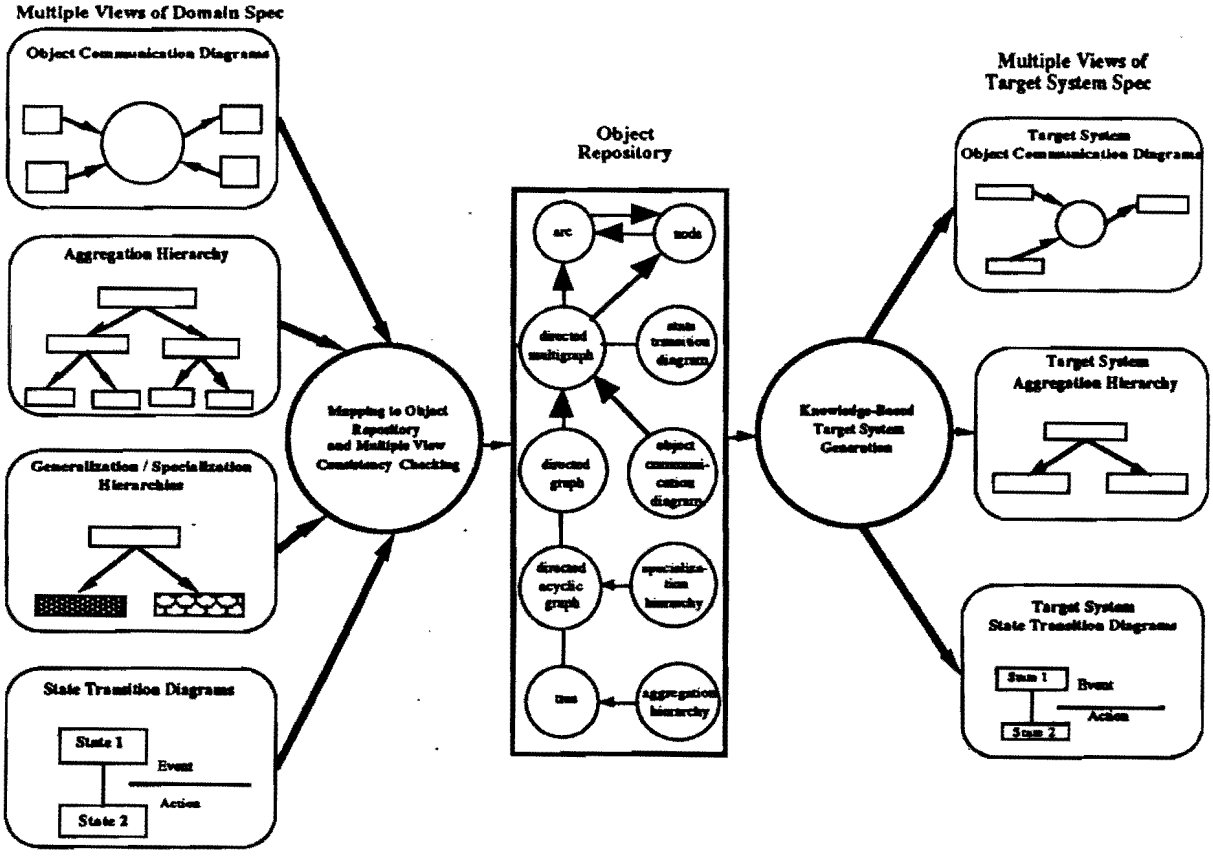

Figure 1: Prototype Domain Modeling Environment

The domain specification stored in the object repository is augmented with domain features (requirements), inter-feature dependencies and feature/object dependencies. Inter-feature dependencies capture the relationships among features. For example, a feature may require the presence of some other feature(s) as prerequisite. Another example of inter-feature dependency is that some features may be mutually exclusive or mutually inclusive with each other. The feature/object dependencies relate features to objects, i.e., they define the object types required to support a particular feature. The domain analyst provides this feature-related information

using the Feature Object Editor. Feature/object dependencies are stored in the object repository.

The object repository provides a unique and consistent specification of the domain model, which can be accessed by various tools. Once the domain modeling activity is completed, the domain specification serves as the framework for generating target systems. The process of generating target systems from a domain model can be substantially improved with knowledge-based tool support. A knowledge-based system called the Knowledge-Based Requirements Elicitation Tool (KBRET) is being developed to automate the process of generating the specifications for target systems from a domain specification. This tool is implemented in NASA's CLIPS expert system shell. It conducts a dialog with the human target system requirements engineer, prompting the engineer for target system specific information. The output of this tool is used to adapt the domain model to generate the target system specification. When the target system objects have been assembled, the corresponding multiple views are derived by tailoring the multiple views of the domain model. The multiple views of the target system are then displayed using StP. The prototype software engineering environment is a domain independent environment. Thus it may be used to support the development of a domain model for any application domain that has been analyzed, and to generate target system specifications from it.

4. KNOWLEDGE-BASED REQUIREMENTS ELICITATION TOOL

A target system specification is derived from the domain model by tailoring it according to the requirements specified for the target system. The process of generating a target system specification consists of gathering the requirements in terms of domain features, retrieving from the domain model the corresponding components to support those features, and reasoning about inter-feature and feature/object dependencies to ensure consistency. The Knowledge-Based Requirements Elicitation Tool (KBRET) facilitates the process of generating target system specifications from a domain model with multiple viewpoints. KBRET is implemented in CLIPS (C Language Integrated Production System), developed at NASA/Johnson Space Center.

The architecture of KBRET consists of two types of knowledge: domain independent and domain dependent knowledge. The domain independent knowledge provides control knowledge for the various functions supported by KBRET. These functions include a browser, a feature selector, a dependency checker, and a target system generator. The domain dependent knowledge represents the multiple views of an application domain model, including the feature/object dependencies. This knowledge is derived from the object repository through the KBRET-Object Repository interface and structured as CLIPS facts (Sugumaran, 1991). The various components of KBRET are diagrammatically depicted in Figure 2.

The separation of domain-independent and domain-dependent knowledge is essential for providing scale-up and maintainability of domain specifications for large domains. Also, since the domain-independent knowledge is independent of the application domain, it can be used with domain-dependent knowledge from any application domain to generate target system specifications for that domain.

4.1. Domain Dependent Knowledge

The domain dependent knowledge sources contain specific information about a particular application domain. They are used by the domain independent knowledge sources of KBRET in eliciting the requirements and generating the target system specification. The domain

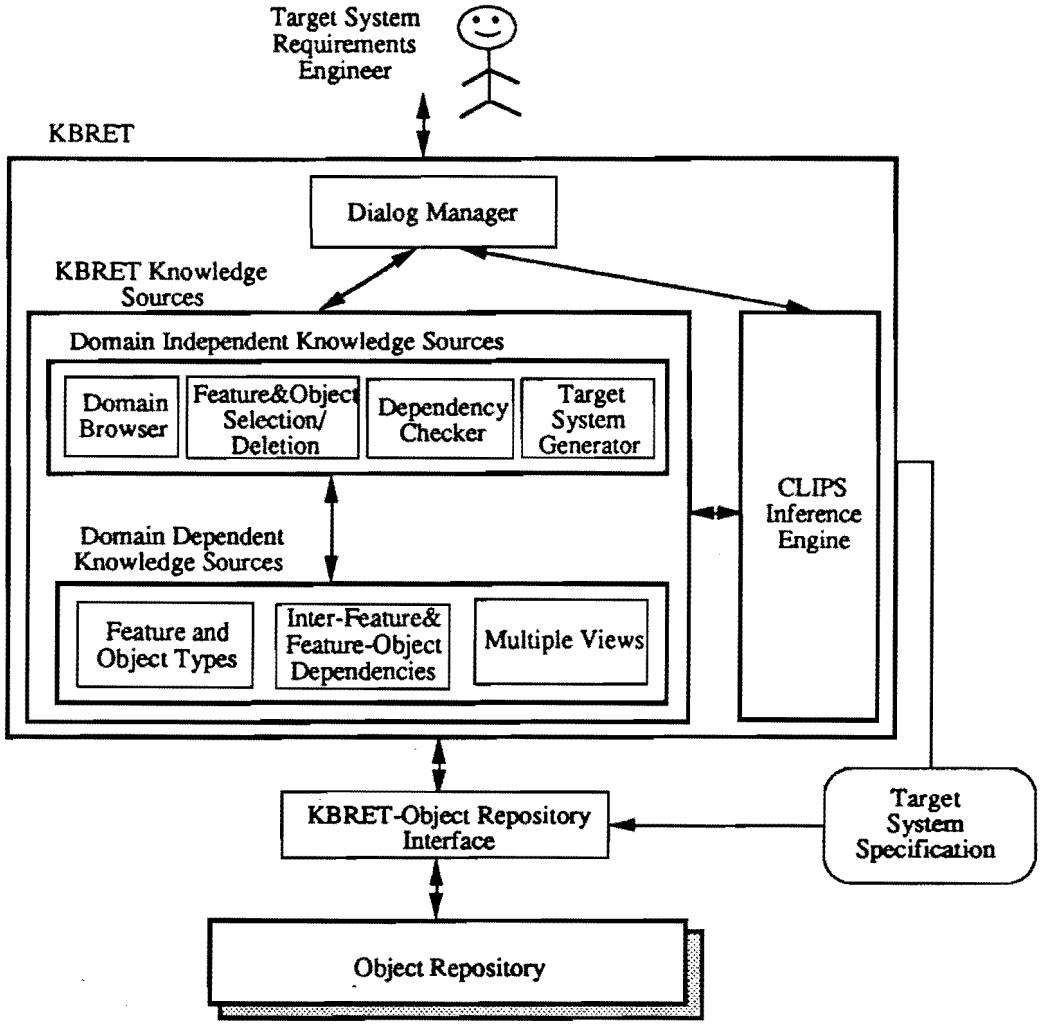

Figure 2. Knowledge Based Requirements Elicitation Tool (KBRET)

dependent knowledge sources are derived from the domain specification, which is persistently stored in the object repository. The KBRET-Object Repository Interface accesses the object repository and creates these knowledge sources using a representation that is compatible with the other knowledge sources of KBRET.

The *Features and Object Types* knowledge source contains a list of all the object types and features that have been incorporated in the domain model. For each object type, its name and properties are stored in this knowledge source. The properties of objects are: kernel, optional, variant, aggregate, agh_root and gsh_root. The CLIPS implementation of this knowledge source is essentially a list of facts - one fact for each object type and its properties and one fact for each feature.

The various relationships and dependencies among features and between features and object types are captured in the *Inter-Feature & Feature-Object Dependencies* knowledge source. The prerequisite relationship between two features is captured in a CLIPS fact with the key word "requires". For each feature, the object types required to support that feature are expressed as CLIPS facts using the key word "supported-by". These dependencies are enforced during feature

selection or deletion by the *Dependency Checker* knowledge source.

The *Multiple Views* knowledge source contains the different views created using the EDLC methodology, in particular, the aggregation hierarchy and the generalization/ specialization hierarchies. These hierarchies are accessed and utilized by the *Target System Generator* knowledge source when the target system is being assembled.

4.2. Domain Independent Knowledge

The domain independent knowledge sources provide procedural and control knowledge for the various functions supported by KBRET. The *Dialog Manager* is responsible for carrying out a meaningful dialog with the target system engineer and elicit the requirements for the target system. It addresses such issues as how, and in what sequence the target system engineer should be prompted for various features, invoking and controlling the different phases of KBRET.

Before specifying the requirements for the target system, the target system engineer may wish to browse through portions of the domain model in order to gain understanding of the application domain under consideration. The *Domain Browser* knowledge source provides this facility. It provides rules for initiating and terminating the browsing facility and also the appropriate domain dependent knowledge sources to be accessed in order to facilitate the browsing of those parts of the domain model which the target system engineer wishes to explore.

The *Feature & Object Selection/Deletion* knowledge source keeps track of the selection or deletion of features for the target system and the corresponding object types. This knowledge source incorporates rules for selecting and deleting features and also invoking the appropriate rules for checking inter-feature and feature/object dependencies.

The *Dependency Checker* knowledge source cooperatively works with the Feature & Object Selection/Deletion knowledge source. When a particular feature is selected for the target system, the *Dependency Checker* enforces the inter-feature and feature/object dependencies for that feature. These dependencies are obtained from the *Inter-Feature & Feature-Object Dependencies* knowledge source which is domain dependent, as shown in Figure 2. When a feature with some prerequisite features is selected, the *Dependency Checker* ensures that those prerequisite features are included in the target system. For example, in the POCC domain, the Verifying Real Time Commands feature requires Sending Real Time Commands feature. If the Sending Real Time Commands feature is not selected and the Verifying Real Time Commands feature is desired in the target system, the Sending Real Time Commands feature will be included in the target system before selecting the Verifying Real Time Commands feature.

Similarly, before deleting a feature from the target system, dependency checking is performed to ensure that it is not required by any other target system feature. Using the example from the previous paragraph, if both Sending Real Time Commands and Verifying Real Time Commands features are selected for the target system, the Sending Real Time Commands feature cannot be deleted from the target system as long as the Verifying Real Time Commands feature is selected for the target system. Thus, the *Dependency Checker* knowledge source has rules to enforce the inter-feature and feature/object dependencies so that a consistent target system is specified.

Once the feature selection for the target system is complete, the *Target System Generator* knowledge source begins the process of assembling the target system. The domain kernel object types are automatically included in the target system. Depending upon the features selected for the target system, the corresponding variant and optional object types are included according to the feature/object dependencies. KBRET also outputs two files containing the target system information. These files are used to tailor the domain model graphical views and generate a set

of graphical views for the target system. The target system views differ from those of the domain model in two ways. First, the optional objects that are not selected for the target system are removed. Secondly, in the case where one or more variants of a domain object type are selected, the object type is replaced by its variant(s).

5. CONCLUSIONS

This paper has discussed the domain modeling approach to software reuse and presented a prototype environment that supports domain modeling and generation of target system specifications. The architecture of the knowledge-based tool, KBRET, along with its domain dependent and independent knowledge sources was described. Finally, the paper has discussed KBRET's approach to generating target system specifications from the domain model, by eliciting the requirements for the target system and tailoring the domain model. Future work includes extending KBRET to provide the capability to generate target system specifications from existing related target system specifications in conjunction with the domain model, and also to improve KBRET's user interface.

6. ACKNOWLEDGEMENTS

We acknowledge the assistance of S. Bailin, R. Dutilly, J. M. Moore, and W. Truszkowski in providing us with information on the POCC. We gratefully acknowledge the major contributions of Liz O'Hara-Schettino, C. Bosch and I. Tavakoli for their major contributions to the prototype software engineering environment. This work was sponsored primarily by NASA Goddard Space Flight Center Automated Technology Branch Code 522.3 under the Code R research program, with additional support from the Virginia Center of Innovative Technology. The Software Through Pictures CASE tool was donated to GMU by Interactive Development Environments (IDE).

7. REFERENCES

Batory, D. (1989). The Genesis Database System Compiler: A Result of Domain Modeling. *Proc. Workshop on Domain Modeling for Software Eng., OOPSLA'89*, New Orleans, LA.

Gomaa, H. (1992). An Object-Oriented Domain Analysis and Modeling Method for Software Reuse. *Proc. Hawaii International Conference on System Sciences*, Hawaii.

Gomaa, H., & Kerschberg, L. (1991a). An Evolutionary Domain Life Cycle Model for Domain Modeling and Target System Generation. *Proc. Workshop on Domain Modeling for Software Engineering, Int. Conf. on Software Engineering*, Austin, TX.

Gomaa, H., Fairley, R., and Kerschberg, L. (1989). Towards an Evolutionary Domain Life Cycle Model. *Proc. Workshop on Domain Modeling for Software Eng., OOPSLA*, New Orleans.

Gomaa, H., Kerschberg, L., Bosch, C., Sugumaran, V., and Tavakoli, I. (1991b). A Prototype Software Engineering Environment for Domain Modeling and Reuse. *Proc. Fourth Annual Workshop on Methods and Tools for Reuse*. Herndon, VA.

Lubars, M. D. (1989). Domain Analysis for Multiple Target Systems. *Proc. Workshop on Domain Modeling for Software Eng., OOPSLA'89*, New Orleans, LA.

Parnas, D. (1979). Designing Software for Ease of Extension and Contraction. *IEEE Transactions on Software Eng.*, Vol. 5 No. 2, pp. 128-137.

Prieto-Diaz, R. (1987). Domain Analysis for Reusability. *Proc. of COMPSAC'87*.

Pyster, A. (1990). The Synthesis Process for Software Development. In R. Thayer and M. Dorfman (Eds.). *System and Software Requirements Engineering*, IEEE CS Press.

Sugumaran, V., Gomaa, H., and Kerschberg, L. (1991). Generating Target System Specifications from a Domain Model Using CLIPS. *Proc. of Second Annual Clips Conference*, Houston.