GENERATING TARGET SYSTEM SPECIFICATIONS FROM A DOMAIN MODEL USING CLIPS

Vijayan Sugumaran, Hassan Gomaa and Larry Kerschberg

Center for Software Systems Engineering
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030

Abstract. The quest for reuse in software engineering is still being pursued and researchers are actively investigating the domain modeling approach to software construction. There are several domain modeling efforts reported in the literature and they all agree that the components that are generated from domain modeling are more conducive to reuse. Once a domain model is created, several target systems can be generated by tailoring the domain model or by evolving the domain model and then tailoring it according to the specified requirements. This paper presents the Evolutionary Domain Life Cycle (EDLC) paradigm in which a domain model is created using multiple views, namely, aggregation hierarchy, generalization/specialization hierarchies, object communication diagrams and state transition diagrams. The architecture of the Knowledge Based Requirements Elicitation Tool (KBRET) which is used to generate target system specifications is also presented. The preliminary version of KBRET is implemented in CLIPS.

1.0 INTRODUCTION

It is widely believed that there is a direct relationship between software reuse and computer software productivity. Researchers are constantly exploring new methods and concepts to improve reuse - a problem that has been solved to a greater degree of success in the computer hardware field. Several models and frameworks have been proposed and discussed in (Biggerstaff and Perlis 89); however, the problem of reuse has not yet been solved satisfactorily. Domain Analysis is a fundamental step towards reuse and is a key factor in the success of reusability. Domain analysis artifacts are more conducive to reuse because they capture the essential objects and functions that characterize the domain.

Parnas initially proposed the idea that it will be more advantageous to build a framework for a family of systems rather than build every individual system from scratch (Parnas 79). He argues that 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. Domain modeling addresses the problem of developing a family of systems in which the traditional system development activities like analysis, specification and design are performed at the application domain level and not at the individual system level.

(Prieto-Diaz 88) states that "In domain analysis, common characteristics from similar systems are generalized, objects and operations common to all systems within the same domain are identified and a model is defined to describe their relationships". Domain analysis also considers the variations among the current systems and must evolve to accommodate unanticipated variations as well.

Thus, a domain model is the representation of the common characteristics and variations among a family of software systems in a given application domain. A computer-based domain model generally captures the static and dynamic aspects of the application domain. The static properties include objects of the domain, attributes of those objects and relationship among them. The dynamic properties include the operations associated with objects and the messages passed between objects. The domain model may also include integrity constraints that express the rules which govern the behavior of objects in the domain.

The primary objective of the domain modeling approach to software construction is to increase reuse, i.e., reuse not only of code modules but also of domain knowledge such as domain requirements, specifications and designs. From the domain model, target systems can be generated by tailoring the domain model, or by a combination of evolving the domain model and then tailoring it. A target system is a member of the family of software systems. 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 systems analysis every time a new target system has to be constructed.

The recent Domain Modeling Workshop held at Austin, Texas, during May 91 and a growing body of literature indicate the emphasis on domain modeling and reuse in software development. Several institutions in industry and academia are pursuing research efforts in domain analysis and domain modeling. At the Center for Software Systems Engineering at George Mason University, we are involved in a software engineering project funded partially by NASA/Goddard Space Flight Center through Computer Technology Associates. In this project, Gomaa et al. have developed a software process model called the Evolutionary Domain Life Cycle (EDLC) model that supports the evolutionary development of families of systems (Gomaa et al. 89). From the EDLC domain model, target system specifications can be generated. We are also developing a knowledge based tool called Knowledge Based Requirements Elicitation Tool (KBRET) that will tailor the domain model and generate target system specifications based on the requirements.

This paper is organized as follows: section 2 provides an overview of the EDLC methodology and the environment, section 3 briefly describes the target system specification generation process, section 4 describes the architecture/design of KBRET, section 5 details the implementation of KBRET in CLIPS and its capabilities, section 6 is summary, section 7 is acknowledgements and section 8 is references.

2.0 THE EVOLUTIONARY DOMAIN LIFE CYCLE MODEL

The Evolutionary Domain Life Cycle (EDLC) Model is a software life cycle model that eliminates the traditional distinction between software development and maintenance (Gomaa and Kerschberg 91a). The various activities within the EDLC paradigm are shown in Figure 1.

Software 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. A complete description of the EDLC methodology and related activities is provided in (Gomaa et al. 89). The EDLC domain model supports the following multiple views:

(a) *Aggregation Hierarchy.* The Aggregation Hierarchy is a composition hierarchy, i.e. it supports the IS-PART-OF relationship. It supports the decomposition of complex aggregate objects (subsystems) into less complex objects eventually leading to simple objects at the leaves of the hierarchy. Alternatively, associated objects may be grouped together into more complex aggregate objects. The Aggregation Hierarchy is an important abstraction concept in domain modeling, since it allows domain modelers to reason about complex aggregate objects instead of two or more simpler objects.

b) *Generalization/Specialization Hierarchies.* The Generalization/Specialization Hierarchy supports the IS-A relationship. With the generalization/specialization classification approach, similar objects are grouped into classes. These objects may have some features in common but they may also have variations between them that need to be expressed. Specialization of a class (object type) can be achieved by means of inheritance, which has been applied very effectively in object oriented programming. Meyer (Meyer 88) has convincingly shown the potential benefits of

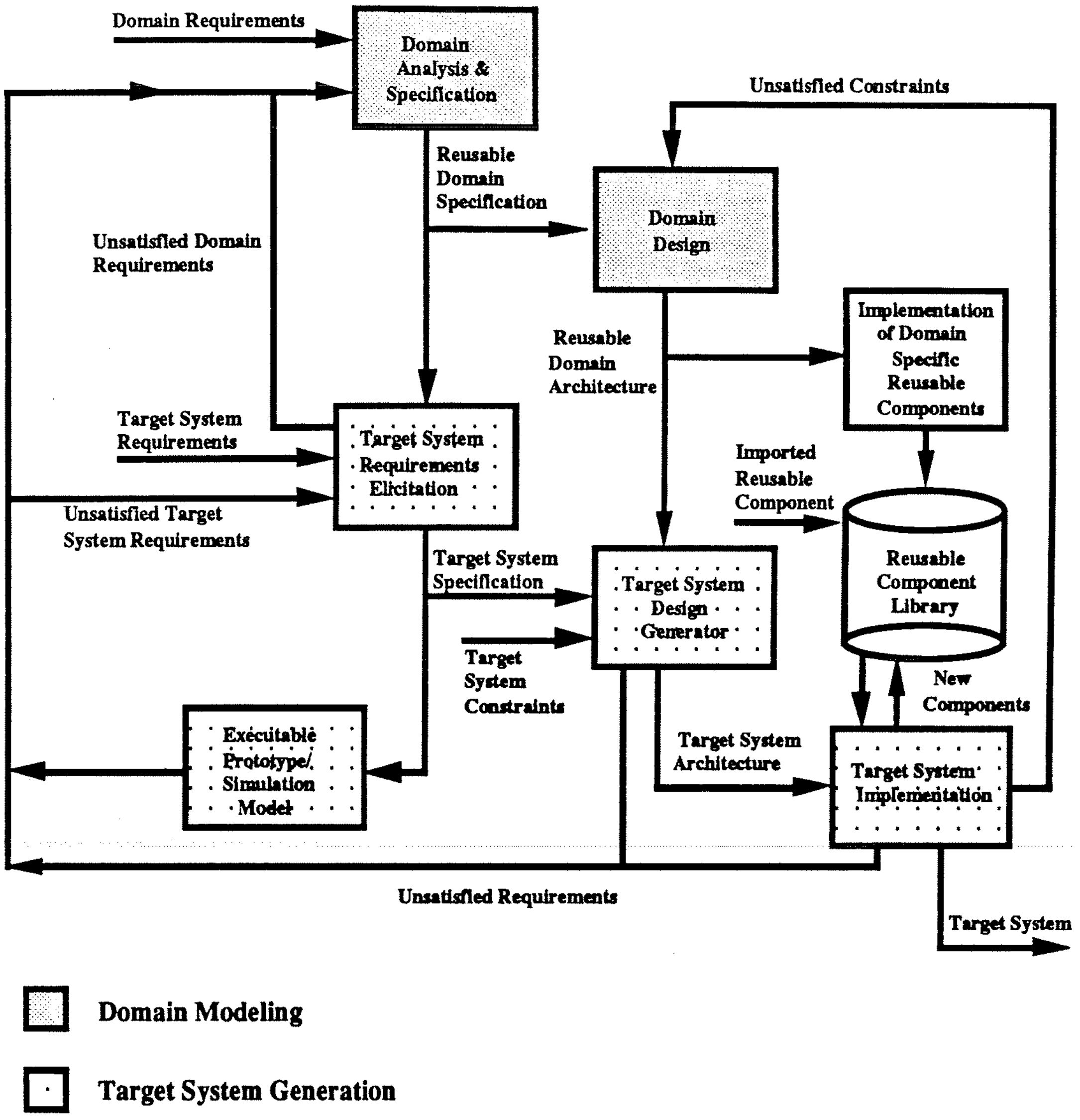
Domain Requirements
Domain Analysis & Specification
Unsatisfied Constraints
Reusable Domain Specification
Domain Design
Unsatisfied Domain Requirements
Reusable Domain Architecture
Implementation of Domain Specific Reusable Components
Target System Requirements
Target System Requirements Elicitation
Unsatisfied Target System Requirements
Imported Reusable Component
Reusable Component Library
Target System Specification
Target System Design Generator
Target System Constraints
New Components
Executable Prototype/ Simulation Model
Target System Architecture
Target System Implementation
Unsatisfied Requirements
Target System
Domain Modeling
Target System Generation

Figure 1. The Evolutionary Domain Life Cycle Model

using inheritance for reuse. Inheritance allows a class to be tailored by adding features, suppressing features or modifying features.

c) *Object Communication Diagrams.* Objects in the real world are modelled as concurrent processes (Jackson 83), which communicate with each other using messages. The object communication diagrams, which are hierarchically structured, show how objects communicate with each other.

d) *State Transition Diagrams.* As each active object is a sequential process, it may be defined by means of a finite state machine and documented using a state transition diagram (although in some cases the finite state machine might be trivial). Each object maintains its own state. An active object supports one operation for each message type that it may receive.

An example of applying the EDLC methodology to an Automobile Cruise Control Problem is given in (Gomaa 90). The domain modeling concepts are currently being applied to NASA's Payload Operations Control Center (POCC) domain at NASA Goddard Space Flight Center. To test these concepts, a domain model has been constructed which captures the similarities and variations of the POCC domain (Gomaa et al. 91b). The following section describes the proof-of-concept demonstration that we are developing for NASA/Goddard.

2.1 EDLC Domain Modeling Environment

A proof-of-concept experiment is under way to develop software tools that support the domain analysis and specification and target system requirements elicitation phases of the EDLC. The experiment uses comercial-of-the-shelf software as well as specially developed software. We are using Software Through Pictures (StP) 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 stored in an object repository.

A knowledge based tool is used to assist with target system requirements elicitation and generation of the target system specification. The tool, implemented in CLIPS, conducts a dialog with the human target system engineer, prompting the engineer for target system specific information. The output of this tool is used to adapt the domain specification to generate the target system specification.

The EDLC domain modeling environment is depicted in Figure 2. In the EDLC paradigm, the domain analyst starts the creation of the domain model by specifying the multiple views. He/she creates the Aggregation Hierarchy, Generalization/Specialization hierarchies, Object Communication Diagrams, and State Transition Diagrams. The domain analyst is not restricted to working with one diagram at a time. A complex system must be understood by a large community of users and these multiple views provide an informal specification of the domain being constructed. Software Through Pictures (StP) is used as the multiple viewpoint graphical editor. StP provides limited form of consistency checking within each view. Additional consistency checking among different views is done by special software that we have developed. Once the graphical views are determined to be consistent, the domain information from these graphical views is extracted from StP's relational database and mapped into the object repository.

The object repository presents an object-oriented representation of the informal specification which can now be enhanced with a formal specification of domain object types, their attributes, their relationship to other object types, the operations associated with the object types, constraints among object types etc. The domain analyst enters this information using the Domain Object Editor. The domain object types are organized in terms of a "meta-schema" that governs the inter-relationships among the objects represented in the multiple viewpoints. This knowledge is used to determine consistency among view points and to evolve the object specification in a consistent manner. The object repository is implemented in Eiffel.

The domain specification stored persistently 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

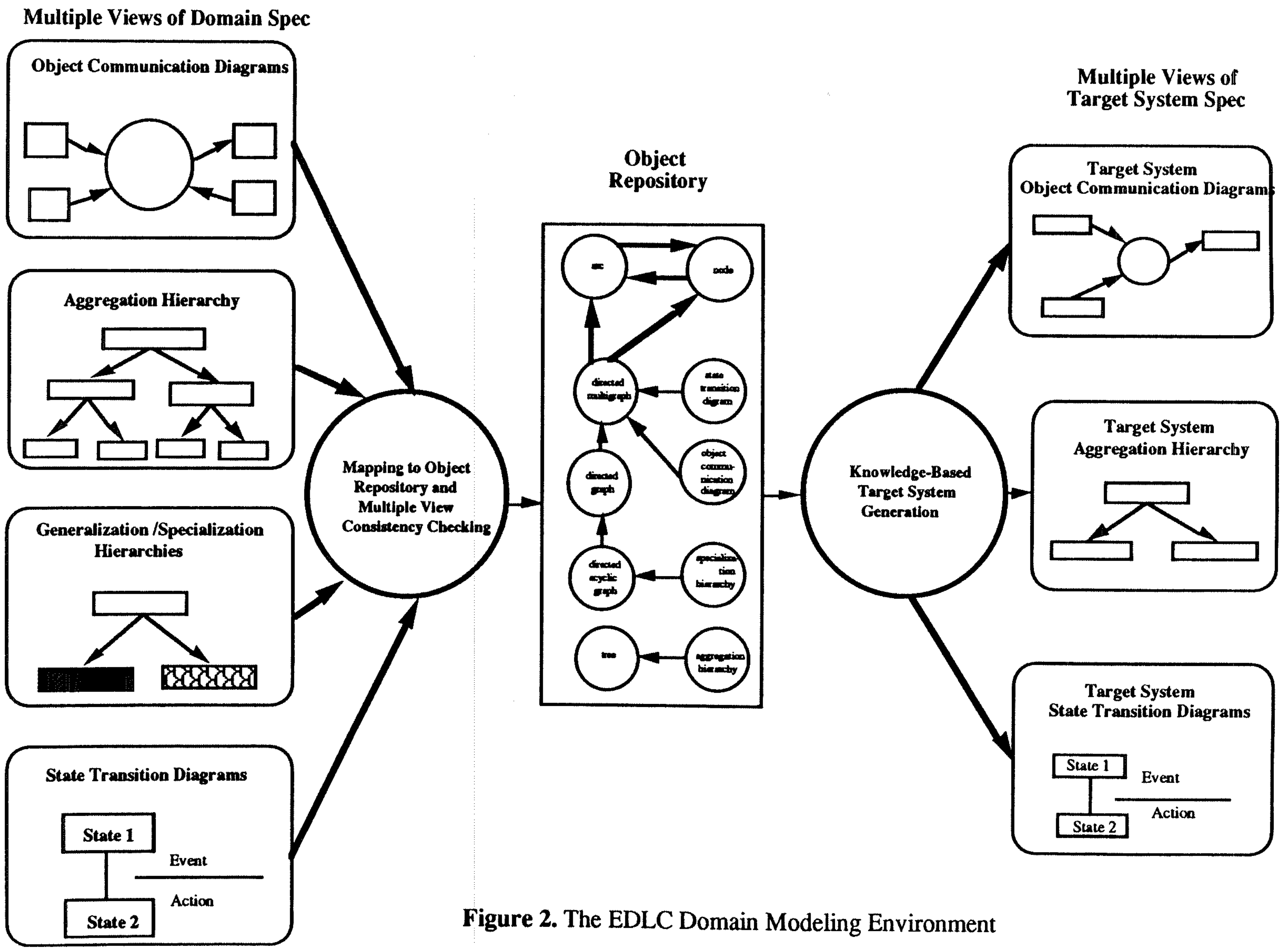
Multiple Views of Domain Spec
Object Communication Diagrams
Aggregation Hierarchy
Generalization /Specialization Hierarchies
State Transition Diagrams
State 1
Event
Action
State 2
Mapping to Object Repository and Multiple View Consistency Checking
Object Repository
arc
node
directed multigraph
state transition diagram
directed graph
object communication diagram
directed acyclic graph
specialization hierarchy
tree
aggregation hierarchy
Knowledge-Based Target System Generation
Multiple Views of Target System Spec
Target System Object Communication Diagrams
Target System Aggregation Hierarchy
Target System State Transition Diagrams
State 1
Event
Action
State 2

Figure 2. The EDLC Domain Modeling Environment

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 and is stored in the object repository as well. The object repository interfaces with knowledge-based tools such as KBRET and provides the informal and formal specifications for reuse. Thus, the object repository provides a unique and consistent specification of the domain model, and this can be accessed by various tools. For example, we have developed a tool that retrieves the domain dependent information from the object repository and creates the domain dependent knowledge sources for KBRET. This tool maps the object repository information into CLIPS facts using the *deffacts* construct.

Once the domain modeling activity is completed, the domain specification serves as the framework for generating target systems. The process of generating target systems calls for knowledge-based tool support. This tool must not only have knowledge about the domain model, but also contain procedural knowledge about constructing target systems. A knowledge-based system called the Knowledge-Based Requirements Elicitation Tool (KBRET) is being developed using CLIPS to automate the process of generating the specifications for target systems. When a target system specification is generated, the corresponding multiple views are then generated by tailoring the domain multiple views and displayed using StP.

The paragraphs above have provided a brief overview of the EDLC methodology and the different tools that are used in creating the domain model and generating the target system specification. The following sections will describe the process of generating target system specification, the knowledge-based tool used in this process, its architecture and implementation.

3.0 TARGET SYSTEM SPECIFICATION GENERATION

The EDLC domain model captures the reusable domain features (requirements) and the dependencies among features and object types. These feature object dependencies provide a powerful indexing mechanism to retrieve reusable components from the domain model, especially object types and their informal and formal specifications. In other words, if a particular domain feature is required in the target system, the object types required to support that feature can be retrieved from the object repository. Thus, the process of generating a target system specification essentially amounts to gathering the requirements in terms of the domain features and retrieving from the domain model the corresponding components to support those features and reason about inter-feature and feature/object dependencies to ensure consistency. This process involves tailoring the domain model and creating the target system specification.

The object types in the EDLC paradigm are classified as kernel, optional, or variant. A kernel object is part of every member in the family of systems. An optional object supports a domain feature and it may or may not be included in the target system. A variant object is a specialization of a kernel or optional object and it also supports a certain domain feature. If multiple variants of the same object type are included in the target system, they have to be integrated to form one synthesized object. However, certain application domains may require the existence of multiple specializations of the same object type, as for example, in the POCC domain, multiple specialized experiments may co-exist on a mission.

Once the requirements have been gathered, the target system can be assembled by including the kernel object types, the selected optional object types and variant object types and integrating the variant object types, if necessary, with the help of the domain analyst. In general, the target system specification generation process consists of the following activities:

(a) accessing and retrieving necessary components from the various knowledge sources;
(b) displaying the information to the target system engineer;
(c) eliciting the target system requirements from the target system engineer;

(d) reason about the inter-feature and feature/object dependencies to ensure consistency;
(e) tailoring the domain specification to generate the target system (this step may or may not require variant integration); and
(f) target system consistency checking.

In order to support the above activities, the target system specification generation tool should be designed in such a way that the target system engineer could browse the domain model, interactively specify the requirements for the target system. The tool should also have the reasoning capability to ensure that a consistent specification for the target system has been generated. The following section describes the architecture of the Knowledge Based Requirements Elicitation Tool (KBRET).

4.0 KNOWLEDGE BASED REQUIREMENTS ELICITATION TOOL (KBRET)

KBRET accomplishes the task of generating the target system specification in several phases. Some of the phases that KBRET may go through are: Browsing, Target System Requirements Elicitation, Dependency Checking, Target System Generation, Variant Object Type Integration.

4.1 KBRET User Interface

The current version of KBRET uses a menu based approach for interacting with the target system engineer. From the main menu, the browsing phase or the target system requirements elicitation phase can be initiated. In the browsing phase, the target system engineer can browse the features captured in the domain model and also get explanations for those features. When the target system requirements elicitation phase is initiated, KBRET provides the domain features in a menu form and the target system engineer can select from this menu the features desired in the target system. Once the features required in the target system are selected, KBRET presents the selected features in a menu form and the target system engineer can delete some of the features selected for the target system.

Whenever a feature is selected or deleted, the dependency checking phase is invoked to ensure consistency. The target system engineer has the flexibility to select or delete features at any time during the target system requirements elicitation phase. When the requirements elicitation phase is exited, the target system construction phase can be initiated by selecting that option from the menu that KBRET provides. This menu also provides an option for specifying features that are not in the domain model. If this option is selected, KBRET suspends the session after gathering information about these new features. This information is provided to the domain analyst to enhance the domain model. When the new features are incorporated in the domain model, the target system engineer can continue the session and finish generating the target system specification. A sample session with KBRET is given in the Appendix.

To support this phased approach, KBRET utilizes various knowledge sources. These knowledge sources can be categorized as domain independent and domain dependent. This separation between the domain-independent and domain-dependent knowledge is essential for providing scale-up and maintainability of domain specifications for large domains. The various components, including the different knowledge sources of KBRET are schematically shown in Figure 3 and discussed in the following sections.

4.2 Domain Independent Knowledge Sources

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

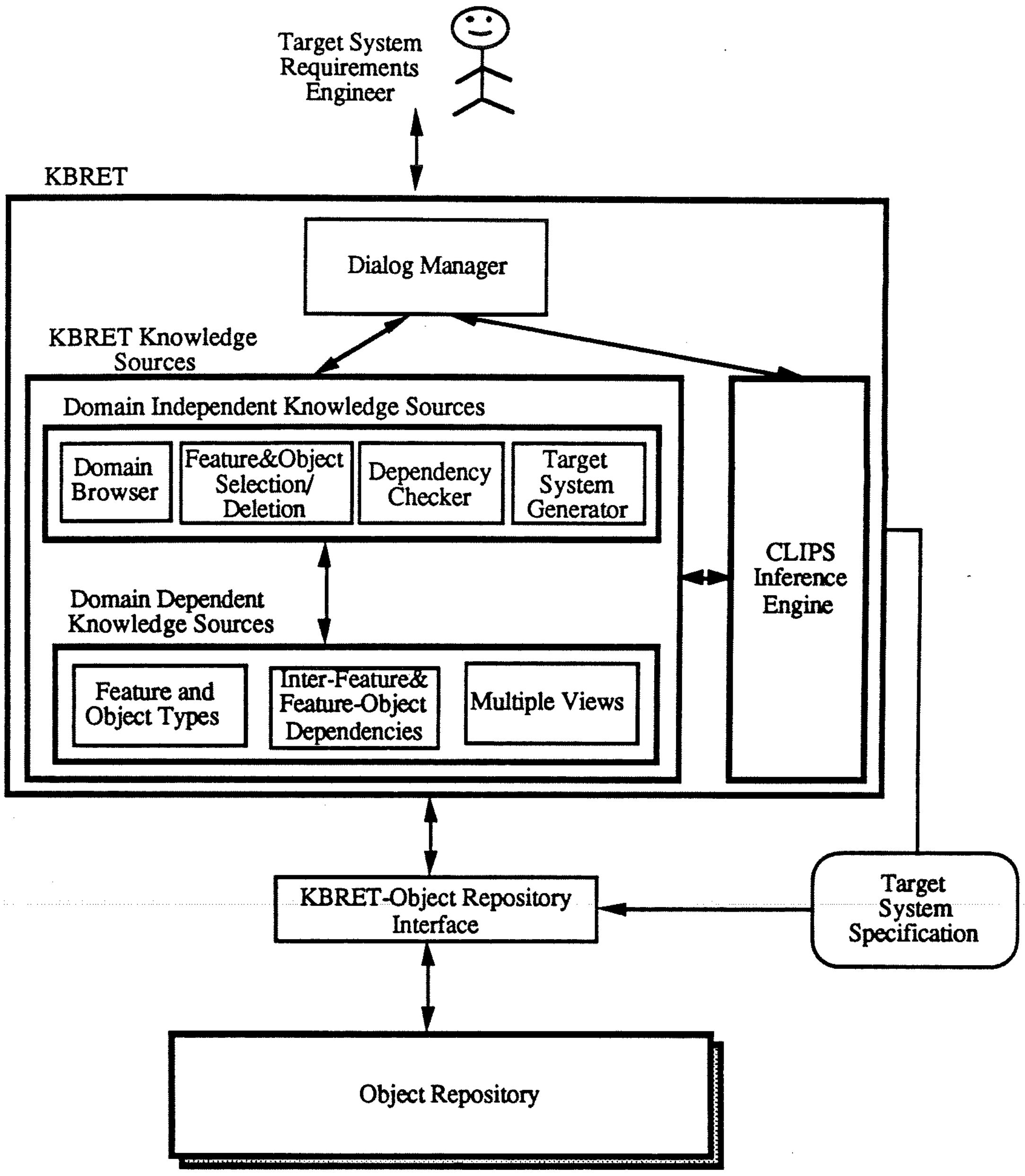

Figure 3. Knowledge Based Requirements Elicitation Tool (KBRET)

prompted for various features, invoking and controlling the different phases of KBRET, the user interface etc.

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 3. 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. The *Target System Generator* would detect if more than one variant (specialization) of a particular kernel or optional object type were included in the target system. These multiple variant object types have to be "integrated" to produce one integrated variant object type that would support the desired features in the target system. Some domains may require the presence of multiple variants of certain objects and those variant objects should not be integrated. For example, in the POCC domain, multiple variants of observatory related objects should not be integrated.

If multiple specializations of a particular kernel or optional object have been selected and if they have to be integrated, the *Target System Generator* would access the *Multiple Views* domain dependent knowledge source and check the appropriate generalization/specialization hierarchy to see if an integrated object type for those variant object types exists as a result of previous variant integration processes. If such an integrated object type is present, then that object type is included in the target system in lieu of those variant object types to be integrated. Once all the required integrated variant object types have been included, the target system generation is complete.

If the integrated variant object type is not present in the domain model, the target system generation process is suspended and the target system engineer is notified about the need for variant integration and the variant object types to be integrated are presented to the domain analyst for integration and enhancing the domain model. Variant integration is a non trivial task and may require considerable domain knowledge. Hence, completely automating the variant integration process will be a tremendous challenge. When the integrated variant object is made available to the *Target System Generator*, the target system generation process is resumed.

4.3 Domain Dependent Knowledge Sources

As the name suggests, 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 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. Some example facts from this knowledge source for the POCC domain are given below:

```
(Object: Command_Load_Processor kernel aggregate)
(Object: Observatory_Instrument_Telemetry_Equation_Processor optional gsh_root)
(Feature: Sending Real Time Commands)
```

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. A few example dependencies from the POCC domain are given below:

```
(Verifying Real Time Commands requires Sending Real Time Commands)
(Verifying Real Time Commands supported-by Earth_Bound_Real-Time_Command Verifier)
```

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. The parent-child relationship between objects in the aggregation hierarchy is expressed as CLIPS facts using the key word "is-part-of". The supertype-subtype relation between objects in the generalization/specialization hierarchy is expressed as CLIPS facts with "is-a" key word. Sample CLIPS facts from this knowledge source are given below:

```
(is-part-of Real-Time_Command_Processor Satellite_Bound_Real-Time_Command_Processor)
(is-a POCC_Mode_Selector POCC_Mode_Selector_With_Simulation)
```

5.0 KBRET IMPLEMENTATION IN CLIPS

A prototype of the Knowledge Based Requirements Elicitation Tool has been developed using CLIPS which is an expert system shell developed by the Artificial Intelligence Section of the Mission Planning and Analysis Division at NASA/Johnson Space Center. CLIPS is written in 'C' and supports backward and forward-chaining, Rete algorithm for pattern matching, wildcards and single and multifield variables, externally defined functions in C or Ada or Fortran. Also, CLIPS can be embedded in application programs written in C, Ada or Fortran.

The basic elements of CLIPS are: fact-base, knowledge-base and inference engine (Giarratano 91). Facts are the basic form of information in a CLIPS system. Rules are fired based on the existence or non existence of facts in the fact-base. A fact is constructed of several positional fields separated by spaces or, a word followed by slots separated by parentheses. Facts may be asserted into the fact-base prior to starting execution and may be added or removed as the action of a rule firing.

A CLIPS knowledge base is represented in the form of production rules. The left hand side (LHS) of a rule is a series of relation or patterns which represent the conditions that must be satisfied for the rule to be fired. Logical operators may be used in the LHS for constraining the patterns. The right hand side (RHS) of the rule is the action to be performed as a result of the rule firing.

The inference engine examines the knowledge base to see if any rule's conditions have been met by searching the fact-base. All rules whose conditions are met are activated and placed in the agenda. The sequence in which these rules are executed is determined by the priorities assigned to those rules. The top rule in the agenda is selected and its RHS actions are executed. As a result of RHS actions, new rules may be activated or deactivated. This cycle is repeated until all rules that can fire have done so or until the rule limit is reached. The number of rule firings allowed in a cycle may be set apriori.

The domain independent knowledge sources of KBRET are implemented as CLIPS rules. The domain specific information contained in the domain dependent knowledge sources are expressed as CLIPS facts. These facts are asserted into the fact-base before the requirements elicitation process begins. When KBRET begins execution, the *Dialog Manager* initiates the dialog with the target system engineer. The rules in this knowledge source are written in such a way that the course of the dialog and the invocation of the different phases are determined by the target system requirements engineer's responses.

At the start of the dialog, KBRET prints the system banner and prompts the target system engineer if he/she wishes to browse the domain model or would like to specify the requirements for the target system, as shown in the sample dialog in the Appendix. If the response is to browse, the browsing phase is initiated. The target system engineer can explore the domain model and get explanations for the different features incorporated in the domain model. Once sufficient familiarity with the domain model has been gained, the target system requirements specification phase may be initiated.

The target system engineer is presented with the various features captured in the domain model in the form of a menu, as shown in the Appendix, and the features desired in the target system can be selected from this menu. Whenever a feature is selected for the target system, the dependency checking phase is initiated and the inter-feature and feature/object dependencies are checked and enforced. If a particular feature, say "F", requires the presence of other features, and they are not selected for the target system, the target system engineer is informed of that fact and those features are automatically included in the target system in order to support feature "F".

An example of this feature dependency checking is shown in the sample dialog in the Appendix. When the target system engineer tries to select the Verifying Real Time Commands (feature 7), KBRET comes back with a message saying that Verifying Real Time Commands requires Sending Real Time Commands feature and it will be automatically included in the target system, and requests the target system engineer's confirmation. When the target system engineer types "y" to confirm the selection, KBRET includes both the Sending Real Time Commands feature and the Verifying Real Time Commands feature in the target system and displays a message to that effect, as shown in the Appendix.

When a feature is selected for the target system, the object types that are required to support that feature are also selected in accordance with the feature/object dependencies and the CLIPS fact-base is updated to reflect that fact. The target system engineer thus, can specify the requirements for the target system and the *Feature & Object Selection/Deletion* knowledge source asserts new facts into the fact-base to record those selections. Of course, the *Dependency Checker* would ensure that the inter-feature and feature/object dependencies have not been violated.

The target system engineer can also delete features that have been selected for the target system. If a feature, say "F", is to be deleted, the *Dependency Checker* will check the fact-base to see if any of the features selected for the target system require that feature "F". If so, the deletion of feature "F" is disabled. An example this deletion dependency checking is shown in the sample dialog. When the target system engineer tries to delete the Sending Real Time Commands (feature 6) from the target system, KBRET comes back with a message saying that the Sending Real Time Commands feature is required by the Verifying Real Time Commands feature and since Verifying Real Time Commands feature is currently selected for the target system, the Sending Real Time Commands feature cannot be deleted, and the dialog continues. When a feature "F" is deleted, it may cause the deletion of some other features if those features were included in the target system solely because of the selection of feature "F" and if they are not required by any other feature selected for the target system. The deletion of a feature also triggers the deletion of object types that were included to support that feature.

If the target system engineer would like to specify a feature that has not been captured in the domain model, the requirements elicitation phase is suspended and the domain analyst is called upon to model that requirement and enhance the domain model. Then, the target system specification and generation may be resumed.

Once the requirements for the target system have been completely specified, the target system generation phase is invoked. KBRET prompts for a name for the target system that is being generated so that it could be stored in the object repository for reuse. The fact-base is examined and the features and the object types selected for the target system are gathered. KBRET then presents the list of features that have been selected for the target system. The kernel object types are included in the target system because they must be part of every member of the family of systems. The selected variant and optional object types are examined to see if variant integration is required. If variant integration is not required, then the target system specification is generated and presented to the target system engineer.

In presenting the target system specification, KBRET provides two options. The target system engineer may view only the leaf level object types or he/she can view both the aggregate and leaf level object types. If the target system engineer chooses the second option, KBRET provides the aggregation hierarchy for the target system, as shown in the Appendix. This is accomplished by pruning the domain aggregation hierarchy, i.e., deleting from the domain aggregation hierarchy the object types that have not been included in the target system. KBRET presents the target system aggregation hierarchy in an indented form, as shown in the Appendix, to reflect the various levels of the aggregation hierarchy.

If variant integration is required, the domain analyst is called upon to perform variant integration. When the integration process is completed, the target system generation phase is resumed and the target system specification is generated and presented to the target system engineer.

6.0 SUMMARY

Domain modeling field is rapidly growing and early results show that domain modeling effectively addresses some of the problems in reuse. We have given an overview of the Evolutionary Domain Life Cycle (EDLC) model and the activities within this paradigm. A domain model of the NASA/Goddard Payload Operations Control Center (POCC) domain is being developed as a proof-of-concept of our EDLC methodology. From the EDLC domain model, several target system specifications can be generated. We have discussed the target system generation process as well as the tools used in accomplishing this task. The architecture of the Knowledge Based Requirements Elicitation Tool (KBRET) and its implementation in CLIPS is also presented. KBRET interacts with the target system engineer, elicits the requirements and generates the target system specification by tailoring the domain specification.

7.0 ACKNOWLEDGEMENTS

We gratefully acknowledge the assistance of S. Bailin, R. Dutilly, J.M. Moore, and W. Truszkowski in providing us with information on the POCC. This work was sponsored by NASA Goddard Space Flight Center with assistance from Virginia Center of Innovative Technology. The Software Through Pictures CASE tool was donated to GMU by Interactive Development Environments.

8.0 REFERENCES

Biggerstaff, T.J, Perlis, A. J., (ed) (1989) *Software Reusability Concepts and Models, Volume I and II*, ACM Press Frontier Series, Addison Wesley.

Giarratano, J. C. (1991). Clips User's Guide, Version 5.0, Software Technology Branch, Lyndon B. Johnson Space Center.

Gomaa, H. (1990). A domain analysis, specification and design method for concurrent systems, George Mason University Report, September.

Gomaa, H., Fairly, R., Kerschberg, L., Sugumaran, V., O'Hara-Schettino, E., and Tavakoli, I., (1989). Sustaining engineering: life cycle support for evolutionary software development, *Research Report Prepared for NASA Goddard Space Flight Center.*

Gomaa, H., Kerschberg, L. (1991a). An evolutionary domain life cycle for domain modeling and target system generation, *Proc. of Domain Modeling Workshop*, May 13, pp. 65-71.

Gomaa, H., Kerschberg, L., Sugumaran, V., O'Hara-Schettino, E., and Tavakoli, I., (1991b). Revised domain model for the payload operations control center (pocc) domain, *Research Report Prepared for NASA Goddard Space Flight Center.*

Jackson, M. (1983). *System Development*, Prentice Hall.

Meyer, B. (1988). *Object-Oriented Software Construction*, Prentice Hall.

Parnas, D. (1979). Designing software for ease of extension and contraction, *IEEE Transactions on Software Engineering*, Vol. 5 No. 2, pp. 128-137.

Appendix. Sample Dialog with KBRET for the POCC domain.

```
***************************************************************
*                                                             *
*   KNOWLEDGE BASED REQUIREMENTS ELICITATION TOOL             *
*                          (KBRET)                            *
*                                                             *
***************************************************************

 Requirements Elicitation for POCC domain
 ************************************
You may browse the features incorporated in the Domain Model, specify the requirements for the
Target System or quit KBRET.
 Choices              Perform
 *******              *******
    1            Browse the Domain Model
    2            Specify requirements for Target System
    3            Quit KBRET
Please type your choice and hit return: 1

         Domain Model Browsing Phase
         ***************************
Please select one of the following choices to continue.
 Choices              Perform
 *******              *******
    1                 Explore the Features
    2                 Exit Browsing Phase
    3                 Quit KBRET
Please type your selection and hit return: 1

         Feature Exploration
         ****************
For the description of a feature, please type its number.
         Choices            Feature to be described
         *******            ******************
            1            Mission Type One
            2            Mission Type Two
            3            Experiment Type One
            4            Experiment Type Two
            5            Data Collection of Simulated Telemetry
            6            Sending Real Time Commands
            7            Verifying Real Time Commands
            e            Exit Browsing Phase
Please type your selection and hit return: 5

Data Collection of Simulated Telemetry:
**********************************
Simulated Telemetry Data can be collected and analyzed.

         Choices            Feature to be described
         *******            ******************
            1            Mission Type One
            2            Mission Type Two
            3            Experiment Type One
            4            Experiment Type Two
            5            Data Collection of Simulated Telemetry
            6            Sending Real Time Commands
            7            Verifying Real Time Commands
            e            Exit Browsing Phase
Please type your selection and hit return: e

Exiting the Browsing Phase........

You may browse the features incorporated in the Domain Model, or specify the requirements for the
Target System or quit KBRET.
        Choices          Perform
        *******          *******
            1            Browse the Domain Model
            2            Specify requirements for Target System
            3            Quit KBRET
Please type your choice and hit return: 2

                    Target System Requirements Elicitation Phase
                    ****************************************
Now, you will be presented with the features incorporated in the Domain Model.  If a feature is desired in the target
system, please type its number and hit return.  Please select one of the following choices to continue.
        Choices          Feature to be selected
        *******          *****************
            1            Mission Type One
            2            Mission Type Two
            3            Experiment Type One
            4            Experiment Type Two
            5            Data Collection of Simulated Telemetry
            6            Sending Real Time Commands
            7            Verifying Real Time Commands
            e            End selecting features
Please type your selection and hit return:  2
                    The Mission Type Two Feature has been selected for the Target System.

Please select one of the following choices to continue.
        Choices          Feature to be selected
        *******          *****************
            3            Experiment Type One
            4            Experiment Type Two
            5            Data Collection of Simulated Telemetry
            6            Sending Real Time Commands
            7            Verifying Real Time Commands
            e            End selecting features
Please type your selection and hit return:  3
                    The Experiment Type One Feature has been selected for the Target System.

Please select one of the following choices to continue.
        Choices          Feature to be selected
        *******          *****************
            4            Experiment Type Two
            5            Data Collection of Simulated Telemetry
            6            Sending Real Time Commands
            7            Verifying Real Time Commands
            e            End selecting features
Please type your selection and hit return:  4
                    The Experiment Type Two Feature has been selected for the Target System.

Please select one of the following choices to continue.
        Choices          Feature to be selected
        *******          *****************
            5            Data Collection of Simulated Telemetry
            6               Sending Real Time Commands
            7               Verifying Real Time Commands
            e               End selecting features
Please type your selection and hit return:  7
                The Verifying Real Time Commands Feature requires Sending Real Time Commands Feature.
                The Sending Real Time Commands Feature will be automatically included if the Verifying Real
                Time Commands Feature is desired.

Please type 'y' to include or 'n' to not include the Sending Real Time Commands Feature. (y/n): y
        The Sending Real Time Commands Feature has been selected for the Target System.
        The Verifying Real Time Commands Feature has been selected for the Target System.

Please select one of the following choices to continue.
        Choices             Feature to be selected
        *******             *****************
            5               Data Collection of Simulated Telemetry
            e               End selecting features
Please type your selection and hit return:  e

Target System feature selection has been exited...

The following features are currently selected:
***********************************
        Number              Feature Name
        *******             ***********
            2               Mission Type Two
            3               Experiment Type One
            4               Experiment Type Two
            6               Sending Real Time Commands
            7               Verifying Real Time Commands

Please select one of the following choices to continue.
        Choices                 Perform
        *******                 *******
            1               Select more features for Target System
            2               Delete a feature from Target System
            3               Specify features not in the Domain Model
            4               Initiate Target System Generation Phase
            5               Quit KBRET
Please type your selection and hit return:  2

Please select one of the following choices to continue.
        Choices             Feature to be deleted
        *******             *****************
            2               Mission Type Two
            3               Experiment Type One
            4               Experiment Type Two
            6               Sending Real Time Commands
            7               Verifying Real Time Commands
            e               End deleting features
Please type your selection and hit return:  6

        Since the Sending Real Time Commands Feature is required by the Verifying Real Time Commands
        Feature and since the Verifying Real Time Commands Feature is currently desired in the Target System, the
        Sending Real Time Commands Feature may not be deleted now.
Please type (c) and hit return to continue: c

Please select one of the following choices to continue.
```

```
          Choices            Feature to be deleted
          *******            *****************
            2               Mission Type Two
            3               Experiment Type One
            4               Experiment Type Two
            6               Sending Real Time Commands
            7               Verifying Real Time Commands
            e               End deleting features
Please type your selection and hit return:  3
          Since Experiment Type One Feature is not required by any other target system feature, it will be deleted
          from the Target System Features.
Please type 'y' to delete or 'n' to abort the deletion of Experiment Type One Feature (y/n) : y
          The Experiment Type One Feature has been deleted from the Target System.

Please select one of the following choices to continue.
          Choices            Feature to be deleted
          *******            *****************
            2               Mission Type Two
            4               Experiment Type Two
            6               Sending Real Time Commands
            7               Verifying Real Time Commands
            e               End deleting features
Please type your selection and hit return:  e

Target System feature deletion has been exited...

The following features are currently selected:
***********************************
          Number             Feature Name
          *******            ***********
            2               Mission Type Two
            4               Experiment Type Two
            6               Sending Real Time Commands
            7               Verifying Real Time Commands

Please select one of the following choices to continue.
          Choices                  Perform
          *******                  *******
            1               Select more features for Target System
            2               Delete a feature from Target System
            3               Specify features not in the Domain Model
            4               Initiate Target System Generation Phase
            5               Quit KBRET
Please type your selection and hit return:  4
          Invoking the Target System Generation Phase.....

                  Target System Generation Phase:
                  ***************************
Please input a name for the Target System: EXAMPLE

                              EXAMPLE Target System Components
                              *********************************
          The following features have been selected for the EXAMPLE Target System
          **************************************************************
                                   Mission Type Two Feature
                                   Experiment Type Two Feature
                                   Sending Real Time Commands Feature
                                   Verifying Real Time Commands Feature
          Assembling the EXAMPLE Target System. Please Wait........

The Target System Object Types have been assembled.  To view those object types included in the Target System,
Please select one of the following choices:
     Choices                  Perform
     *******                  *******
        1            View Leaf Level Object Types
        2            View Aggregate and Leaf Level Object Types
Please type your selection and hit return: 2

The Aggregate and Leaf Level Objects of the Target System:
************************************************
 Payload Operations Control Center Domain (kernel aggregate)
     Telemetry (kernel aggregate)
          Telemetry Pre-Processor (kernel)
          Spacecraft Telemetry Processor (kernel aggregate)
               Mission Two SC Eng. Telemetry Analog Limits Checker With Eqn. Processing (variant)
               Mission Two SC Engineering Telemetry Trend Analyzer (variant)
               Mission Two SC Engineering Telemetry Equation Processor (variant)
               Mission Two Discrete SC Engineering Telemetry Analyzer (variant)
               Mission Two FDF Interface (variant)
          Observatory Telemetry Processor (kernel aggregate)
               Experiment Two Instrument Telemetry Analog Limits Checker (variant)
               Experiment Two Instrument Telemetry Trend Analyzer (variant)
               Experiment Two Discrete Instrument Telemetry Analyzer (variant)
               Experiment Two Scientific Telemetry Analyzer (variant)
          TAC Interface (kernel)
          RUPS Interface (kernel)
     Command (kernel aggregate)
          Command Load Processor (kernel aggregate)
               Satellite Bound Command Load Processor (kernel)
               Earth Bound Command Load Verifier (kernel)
               Command Load Data Store (kernel)
               OBC Image Verifier (kernel)
               CMS Interface (kernel)
          Real Time Command Processor (optional aggregate)
               Satellite Bound Real-Time Command Processor (optional)
               Earth Bound Real-Time Command Verifier (optional)
               Real-Time Command Data Store (optional)
          Satellite Bound Command Problem Resolver (optional)
     Flight Operations Analyst (kernel aggregate)
          FOA STOL Interface (kernel)
          POCC Mode Selector (kernel)
          FOA Command Processor (kernel)
          FOA NCC Processor (kernel)
          FOA Telemetry Processor (kernel)
          NCC Interface (kernel)
     History (kernel aggregate)
          Telemetry History (kernel)
          Command History (kernel)
          Flight Operations Analyst History (kernel)
          Telemetry Block History (kernel)

The EXAMPLE Target System Generation is complete.  The object types shown above have been included in it and
no variant integration is required.
```