# PRODUCT-LINE REUSE FOR ADA SYSTEMS

James R. Hamilton, Harold G. Hawley, and Clinton J. Lalum
Boeing Defense & Space Group
P. O. Box 3999, M/S 87-37, Seattle, WA 98124-2499 USA

## ABSTRACT

This paper describes how product-line reuse for Ada systems can be provided through integration of object-oriented and rule-based technologies. The Advanced Research Projects Agency (ARPA) Software Technology for Adaptable, Reliable Systems (STARS)/ Navy demonstration project is using a product-line approach to system development.

A STARS product-line approach to system development includes a two life-cycle process where assets are developed for reuse during a domain engineering life-cycle, and subsequent application engineering life-cycles use these reusable assets to generate applications to meet the specific needs of a customer.

The Boeing STARS team has developed an object-oriented software engineering environment (SEE) which supports this product-line reuse paradigm. The SEE utilizes rule-based adaptation and other advanced techniques to provide rapid generation of applications conforming to customer requirements.

## KEYWORDS

Application Engineering
Application Modeling
Asset Retrieval and Adaptation
Domain Engineering
Domain-specific
Object-oriented
Product-line
Rule-based Adaptation
Software Reuse

## 1.0 INTRODUCTION

This paper describes an approach for reuse within the context of an organization which has or is developing a product-line for which a business case analysis has shown good potential for a significant return on investment. That is, the organization has adequate domain expertise and has determined that the investment required to implement a reuse program will result in a return on investment sufficient to justify the assessed risks.

More specifically, the domain example we reference is the Air Vehicle Training System (AVTS) domain, and the approach to software reuse described is a tailored version of the Software Productivity Consortium's (SPC) Reuse-Driven Software Process (RSP). The organizational context is the Naval Air Warfare Command Training Systems Division (NAWCTSD) in Orlando, Florida. For this demonstration program, Boeing has, under contract with the Advanced Research Projects Agency (ARPA) Software Technology for Adaptable, Reliable System (STARS) Program, developed a Software Engineering Environment (SEE) which provides automation support for reuse within this organizational context.

The RSP process employed on the demonstration project is the leveraged version of RSP. The RSP describes a range of reuse approaches which can be used by an organization to achieve their reuse goals. The reuse approaches range from opportunistic to anticipating. The Navy project selected a leveraged approach which is one of the highest levels of reuse described by RSP. Opportunistic reuse can be characterized as developing a general purpose library of parts which can be used by application developers as they see fit. A leveraged reuse approach includes development of an architecture for the product-line's family of systems, well defined and repeatable processes, and generation technologies which insulate the application engineer from the implementation details contained in the reusable assets.

This leveraged approach to application engineering enables an organization to have application engineers that understand business issues and customer needs without needing to know all details of the implementation. In leveraged RSP, application engineers are presented a set of questions to answer, and the resultant answers are used by the generation technologies to produce an application from the architecture-based reusable assets. This is a considerably different approach than the typical opportunistic reuse paradigm where users must browse through Ada code in a reuse library looking for assets they can reuse.

To support leveraged reuse, automation technology must support capturing architecture-based reusable assets and must provide generation technologies for producing specific applications. The SEE automation, used to support the AVTS product-line development, may be viewed as either domain-independent (able to support a variety of domains) or domain-specific (dedicated to the AVTS domain). The viewpoint is dependent on what the SEE is considered to comprise. The "core" SEE (excluding, for example, domain-specific processes, etc.) is domain-independent, whereas the

ACM 0-89791-705-7/95/0011--0341

"extended" SEE as delivered and used in the Navy demonstration project is domain-specific. Throughout the remainder of this paper, "SEE" refers to the "extended" SEE.

The following sections describe the product-line process, some of the technical components of the chosen approach to software reuse, and conclusions based on experience to date.

## 2.0. A PRODUCT-LINE PROCESS

The highest conceptual level of the product-line process which Boeing selected for demonstrating reuse capabilities can be described as being process-driven, domain-specific, reuse-based, and automation-supported. The conceptual foundations of the STARS interpretation of "process-driven" are described in [1], and the STARS vision of "domain specific, reuse-based" is described in [2, 3]. A summary description is that software development is predicated upon well-defined, repeatable processes wherein new systems within a given domain are built largely, if not entirely, from existing parts constructed specifically to be reused, and that automation support is available and utilized for most, if not all, of the overall process.

## 2.1. TWO LIFE-CYCLE PROCESS

Another principle of the STARS vision embodied in this product-line process is the two life-cycle process of domain engineering (DE) and application engineering (AE) depicted in figure 1. The first life-cycle, DE, is an on-going effort which creates (and/or acquires) and maintains the domain architecture and reusable components. The second life-cycle, AE, is repeated for each instance of the product-line (family) which is created.

From an organization's business perspective, DE represents investment, and AE is the vehicle for achieving the desired return on investment. For this reason, effective communication across life-cycle boundaries (indicated by the vertical arrows in figure 1) is a critical component of the overall product-line process.

## 2.2. REUSE-DRIVEN SOFTWARE PROCESS (RSP)

RSP as developed by the SPC is described in detail in [13]. RSP concentrates on the technical issues of both DE and AE, but also includes non-technical (e.g., business) issues of early DE work associated with assessing the viability of a given product-line within the context of a given organization. We do not attempt to make a clear distinction between "product-line" and "domain," except to note that a product-line may comprise one or more domains.

One of the DE products defined by RSP is a "question-decision model" over the product-line's problem-solution spaces. This decision model (DM) is used by an application development project to select a set of reusable components that can be used to produce a specific application instance.

The DM is organized hierarchically (i.e., as a question-decision tree) where each node (decision group) includes a manageable number of questions. During AE, the questions are asked and answered, one decision group at a time, beginning at the top of the hierarchy and proceeding down to the lowest ("leaf") level. At each step, the answers provided may determine what questions are asked next and may constrain allowable answers to subsequent questions.

The objective of this question-answer process is to identify/define a tailored application (solution) based largely (ideally, entirely) on a high-level of abstraction. By answering the questions, the application developer provides the SEE with the information required to generate a solution. This solution will comprise both common and variable parts from the set of reusable components in the domain architecture, where some components are used without modification and others are adapted (tailored) specifically for the given application instance. That is, the application consists of reused parts which have been selected specifically for the application (possibly multiple times), some of which have also been adapted specifically for the application. An approach for meeting the objective of this question-answer process is described in [19, 20].

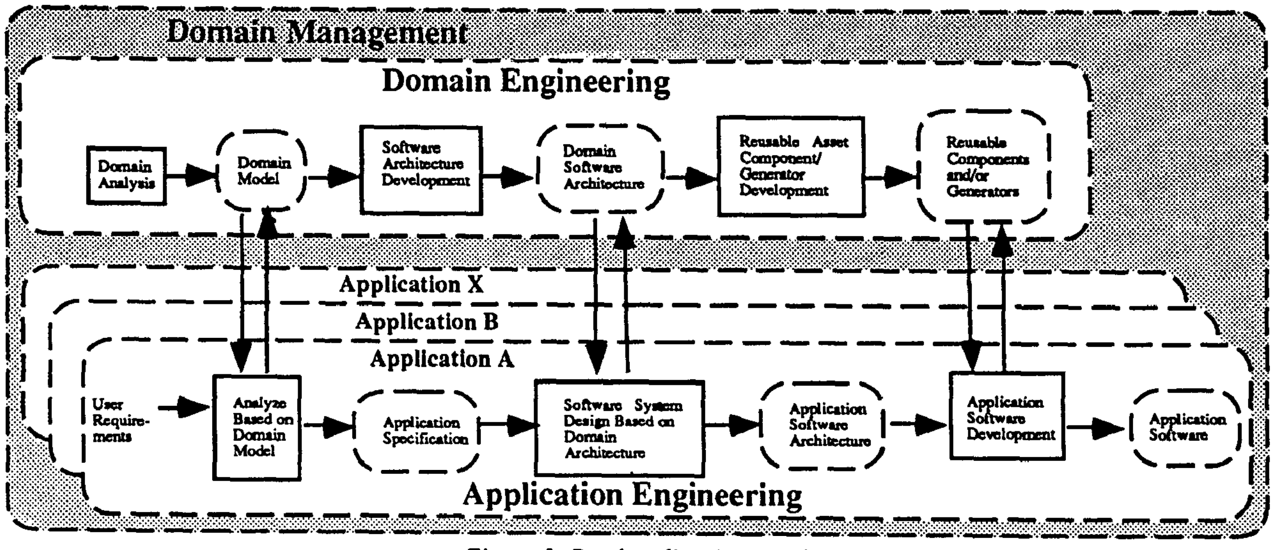
Domain Management
Domain Engineering
Domain Analysis
Domain Model
Software Architecture Development
Domain Software Architecture
Reusable Asset Component/ Generator Development
Reusable Components and/or Generators
Application X
Application B
Application A
User Require-ments
Analyze Based on Domain Model
Application Specification
Software System Design Based on Domain Architecture
Application Software Architecture
Application Software Development
Application Software
Application Engineering

*Figure 1: Product-line Approach*

## 2.3. TAILORED RSP

A tailored version of RSP was developed for use in the Navy demonstration project within the AVTS product-line. The Boeing SEE, developed to provide automated support for this tailored RSP process, includes process definition and automation, object-oriented DE and AE repositories, and an integrated set of tools. Each of these integrated SEE components is either a commercial-off-the-shelf (COTS) product or Boeing-developed software, available through government sources. The tailored RSP process relies on the two life-cycle approach (DE and AE) described earlier. The demonstration project personnel are performing the DE and AE tasks for the Navy demonstration project, and they have developed many refinements to tailor the RSP process. The scope of the demonstration project is confined to the Flight Dynamics domain of the AVTS product-line.

The remainder of this subsection describes those parts of the tailored RSP used in Navy demonstration project which are relevant to the knowledge-based selection and adaptation techniques discussed in the next section. Note: Throughout the remainder of this paper, "RSP" refers to tailored, leveraged RSP as used on the demonstration program unless specifically noted otherwise.

RSP utilizes the capability of the SEE's object-oriented information model (IM) to capture and model fine grained objects and relationships between objects. In particular, answers to questions are modeled as "answer variable" (AV) objects. Instances of another type of object, "decision variable" (DV), are computed from AV's. DV's are thus removed from the actual AE decisions by one level of abstraction. The relationship of DV's to AV's can be characterized as:

$\{DV_n\} = \{p_{n,j}\}(\{AV_j\})$, where

- $\{DV_n\}$ is a set of one or more DV's,
- $\{AV_j\}$ is a set of one or more AV's,
- $\{p_{n,j}\}$ is a set of functions which computes the values of the DV's in $\{DV_n\}$.

Each function p in $\{p_{n,j}\}$ computes the value of a single DV in $\{DV_n\}$, and is modeled as an "expression" object in the repository. The expression (over one or more AV's) is evaluated at run-time to compute the value of the corresponding DV. A single AV may be used to compute multiple DV's (via multiple functions), and multiple AV's may be required to compute a single DV. In effect, the former is an example of deductive reasoning, and the latter a form of inductive reasoning.

A third type of object, the "instantiation parameter" (IP), is utilized for both selection of specific reusable components from available domain assets, and for adaptation of individual selected components. Each IP is computed via an expression (over one or more DV's) in a manner essentially identical to the computation of DV's from AV's. It would be equally viable to compute IP's directly from AV's. The described two-step process was selected in part to reduce complexity in the expressions (functions) used to perform the run-time evaluations.

The functions (expressions) are part of the DE knowledge base. In the initial SEE implementation this knowledge was maintained in a rule-based application [19,20], and all inferences over the knowledge base were performed by this application. The next refinement of the SEE models this knowledge base as repository objects, and distributes portions of the inferences over this knowledge base.

It is important to note that the AE repository user (application engineer) is allowed "read" but not "write" access to the Reuse (DE) repository. The above description of the computation of DV's and IP's omitted reference to specific repositories (AE vs. DE). In reality, the AV, DV, IP, and expression (function) objects are defined in the DE repository, but the computed values are kept strictly in the AE repository.

## 3.0 IMPLEMENTING PRODUCT-LINE REUSE

The implementation of product-line reuse via RSP requires automation to support generating an application. The Boeing SEE supports integral engineering processes (e.g., code development, change and project management, and requirements analysis) which are primarily used during domain engineering, and generation processes which are primarily used during application engineering. The generation techniques being used by the Navy demonstration project make use of the SEE's inherent methods, knowledge base, and integrated tools. The SEEs' IM is primarily comprised of a hierarchy of types with properties and methods. The methods function separately or in conjunction with the integrated tools [21]. The IM has a number of meta data types, properties, methods, and relations provided by the COTS vendor, as well as Boeing extensions to the meta data types.

The two life-cycle approach to RSP centers around the two hierarchical structures described below, the Decision Model (DM) in the DE repository, and the Application Model (AM) in the AE repository.

The DM is created by domain engineers (DE) during the first life-cycle. The DM is a hierarchical structure with a top node under which all design groups created by the DE are attached. It is possible to support multiple AM hierarchies during AE, but a single DM is typically created for each domain. Figure 2 illustrates a typical DM hierarchy for the AVTS domain.

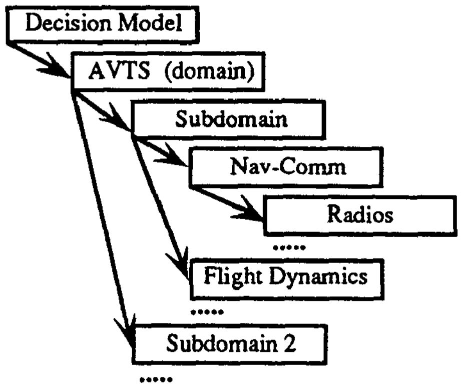

*Figure 2: AVTS Decision Model*

The initialization process creates an AE repository and relates it to the Reuse, i.e. DE repository. The initial starting configuration of the AM within the AE repository is an empty top AM node The structure of the AM is dynamically built by the knowledge base engine in response to answers provided by an application engineer. Figure 3 shows an AM after a user has answered questions to meet a specific set of requirements from the DM in figure 2.

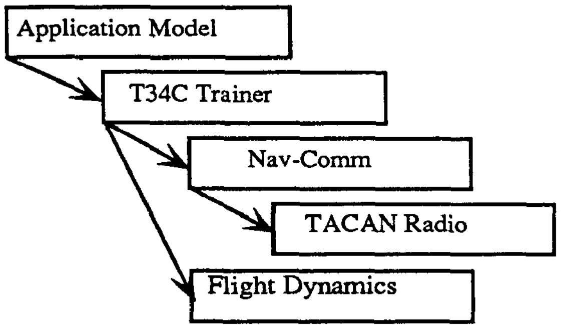

*Figure 3: Application Model*

## 3.1. RSP DOMAIN ENGINEERING OVERVIEW

For the DE life-cycle, there are a number of activities explicitly defining the steps the domain engineer takes in designing, building, and populating the DE repository. The populating also includes the development of the DM.

This paper briefly summarizes some of the activities performed during domain engineering: Decision Model, Product Architecture, Component Design, Generation Design, Component Implementation, and Generation Implementation (sequentially listed in the order performed, as illustrated in figure 4).

The following paragraphs describe each of these processes.

### 3.1.1. DECISION MODEL ACTIVITY

The Decision Model Activity defines the set of decisions an application engineer must resolve to describe and construct a deliverable product. These decisions, and the logical relationships among them, determine the variety of products in the domain. To construct a product, these decisions must be sufficient to distinguish the desired product from all other members of the family. The decisions affect how work products of AE, including Ada source code and documentation, will vary in form and content.

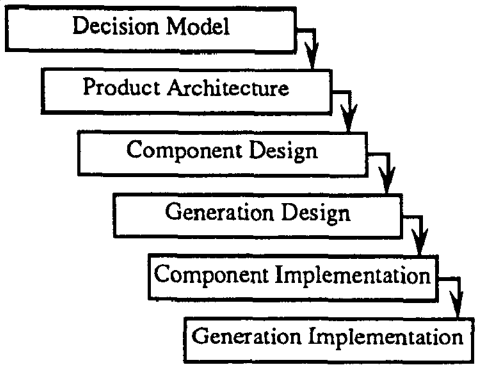

*Figure 4: Selected DE Processes*

In the Decision Model Activity (process), domain engineers elaborate the essential requirements and engineering decisions from the domain assumptions (not discussed here) into an engineering work product identified as the DM. The DM is composed of Decision Tables (DT) which contain DV's. DT's are created in order to provide logical groupings of variation in the product-line. Domain engineers generally create tables for each area of expertise in the domain such as flight dynamics or navigation/communication.

The domain engineers may develop two views of variation in the product-line. One view for domain engineering activities and one view for application engineering activities. Two variation models may be created in some product-lines where a higher level of abstraction is provided for the application engineer and a detailed model is developed for the domain engineers.

The AV's, DV's, and IP's, discussed earlier in section 2.3, are embedded in the DM. The expressions entered into the AV's are evaluated during application engineering (section 3.4.1) to assign values to DV's, and hence to IP's, and thereby identify specific assets and any adaptations applicable to them.

### 3.1.2. PRODUCT ARCHITECTURE ACTIVITY

The objective of the Product Architecture Activity is to define an adaptable architecture for products that can be produced in AE. The key to producing this architecture is providing the flexibility such that this architecture will support all instances of the product family. During Product Architecture, domain engineers identify the structure of each AE work product family in terms of components that application engineer's may produce from adaptable components. Application engineers subsequently create work products by selecting, adapting, and composing instances of adaptable components produced by DE.

### 3.1.3. COMPONENT DESIGN ACTIVITY

The Component Design Activity uses the (previously designed) product architecture to identify the set of adaptable components required to construct an application within the product family. These components are specified and designed in accordance with the previously defined architecture. A set of component designs defines a library of adaptable components that may be adapted and composed to construct applications within the product family.

A component design consists of three parts: Adaptation Specification, Interface Specification, and Functionality Specification. Component Design is required for adaptable and non-adaptable components. Component Design for documentation and other work products (non-code) may or may not include an adaptation specification.

The Adaptation Specification, for an adaptable component, describes the ways that the component can be tailored via a set of parameters. Each parameter has a name and type to indicate its range of variations.

The Interface Specification describes the desired characteristics of the implementation of the component. The form and content of the Interface Specification is particular to the component type and the design method used.

The Functionality Specification is an additional requirement for Component Design that is not an original part of RSP. This specification contains pertinent design information. This design information might include algorithms, control structures, control loop diagrams, etc. It can be parameterized with respect to the variations in the Adaptation Specification.

### 3.1.4. GENERATION DESIGN ACTIVITY

A Generation Design Activity specifies how to select and adapt components according to decisions in the AM and to compose them according to the internal organization of that work product in the Product Architecture. The Generation Design Activity uses the Decision Model, Product Architecture and Component Design

Generation design produces Decision Mappings, Architecture Mappings, and Component Mappings. These mappings are, in effect, the functions described earlier in section 2.3.

Decision Mapping is represented as a pairing between an IP and a corresponding expression. The expression, which is evaluated during application engineering to determine the value of an IP, is described in terms of decisions in the product family's DM. These expressions are part of the domain knowledge base, and are a logical equivalent of rules in a rule-based expert system. The expression syntax is rich, and may involve iteration over a group of decisions or conditional testing of one or more decisions.

The Architecture mapping representation is the same as the Decision mapping, except that the IP's come from the Product Architecture of the work product. Each expression in the Architecture mapping evaluates to TRUE or FALSE, representing the inclusion or exclusion of an architecture node in the resulting application.

The Component mapping representation is the same as the Decision mapping, except that the IP's evaluate to component names (defined in the Component Design). Component mapping determines which components are used in each Product Architecture node included via the Architecture mapping.

### 3.1.5. COMPONENT IMPLEMENTATION ACTIVITY

The objective of Component Implementation is to implement adaptable components that satisfy their respective component designs consistently with respect to product requirements and product architecture. The result of Component Implementation is a complete set of adaptable components that can be used during AE to construct applications or associated work products.

A given component may be anything required to build an application: Ada source code, documentation, or verification/validation data. Such a component, because it is adaptable within a defined range of variation, actually represents a family of (application) components. This variability of adaptable components is an essential factor in enabling application engineers to construct distinct applications within the product family.

For each component, the Component Implementation contains an Adaptation Specification which includes IP's and their constraints. In the case of some common components, there may be no variation, and in such cases the Adaptation Specification is null.

An adaptable Ada component is uniquely named and consists of two parts: an adaptability interface and a body. A component family is characterized by a set of common capabilities and variations in those capabilities. The adaptability interface is a specification of a set of adaptation parameters that provide for the characterization and extraction of a particular instance of a component family. The body is the sum of the potential implementations of all of the components in the family. The term "potential" is used because the parameters are sufficient to select any component family instance uniquely, but the particular implementation either may not be available or may be extracted from a representation of the family or relevant subfamily. This varies with the mechanism used for implementing adaptation of the adaptable component.

### 3.1.6. GENERATION IMPLEMENTATION ACTIVITY

The Generation Implementation Activity produces a Generation Procedure which allows application engineers to produce (generate) a work product in its final form (a set of instantiated components) and associated product documentation from an instance of the DM (aka. AM).

The Generation Procedure has four major portions. The first portion is an implementation of the expressions which map DV's to IP's.

The second is an implementation of the use of IP's to select nodes which will make up the architecture of the final work product and the nodes' corresponding adaptable component.

The third is an implementation of the use of DV's to select the artifacts associated with the component which may be requirements, design, or test items.

The final part of the generation procedure uses IP's to adapt the components selected in step two or the texts selected in step three. An IP (in the DE repository) may either select or adapt (or both) one or more reusable components (in the DE repository), depending on the corresponding IP value (in the AE repository).

The application engineer subsequently answers the questions associated with each design group in the Decision Model (hierarchy of design groups), and thereby generates the AV values. These AV values are used to compute the DV values, which are in turn used to compute the IP values. Finally, the IP values determine which reusable components need to be retrieved from the DE repository and precisely how they are to be adapted for the specific application. This retrieval specification is contained in a retrieve file which is processed via the retrieve method. Section 3.4 provides more details of this selection and adaptation process.

## 3.2. LIMITATIONS OF ADA GENERICS

The creation of adaptable components as discussed in section 3.1.5 allows the design and creation of Ada generic packages (for that matter, any package) but also provides more kinds of instantiation adaptation than is provided in the Ada language. Although Ada provides generic packages as a standard facility that can be used to implement adaptation of source code, it requires explicit code segments for every possible combination that can be instantiated. Adaptation is limited to simple substitution which must be coded into source code instances. Using Ada generics can be used to support simple forms of variation, but it doesn't handle the more complex adaptation techniques such as alternate implementations, conditional expressions/inclusions, multiple instantiations of code units, or adaptation of other types of assets such as requirements and test cases.

The adaptation capabilities that we have developed support all of the above mentioned types of adaptation including adapting assets such as process definitions, requirements, plans, documents, and test data. An example of an adaptable Ada code fragment can be seen in figure 5.

```
procedure $P_PROCEDURE_TEST_NAME$ (
   TEST_ACTIVATE : in BASE_TYPES.DISCRETE_STATE;
      POWER : in BASE_TYPES.SIM_BOOLEAN;
      NEW_POWER : in SIM_BOOLEAN;
      - -$- -Include if change in channel or
      - -$- - frequency terminates self test.
      - -$IF $P_TERM_WITH_CHAN$ THEN
      CHANGE_IN_CHANNEL  : BASE_TYPES.SIM_BOOLEAN;
      - -$END IF
      . . . .
begin
   PHASE_FOUND := false;
   DETERMINED_ACTIVE := false;
   if POWER = True then
      - -$- - Include if type is push and release
      - -$IF $P_INITIATE_PUSH_AND_RELEASE$ THEN
      if TEST_ACTIVATE = On then
         DETERMINED_ACTIVE := true;
         - -$- - Include if btn press restarts test
         - -$- - Iif test is already running.
         - -$IF $P_AFFECT_OF_REPRESS_RESTART$ THEN
         if LAST_PASS_ACTIVE then
            TIMER := 0.0;
         end if;
         - -$END IF
      end if;
      - -$END IF
  . . . .
end $P_PROCEDURE_TEST_NAME$;
```

*Figure 5: Ada Code Fragment*

This adaptation technology has similarities with the C language preprocessor, but it has a more powerful expression capability to support our inference engine. The adaptation language is documented as a BNF grammar and can be easily parsed (e.g., by using YACC and LEX).

## 3.3. IMPLEMENTING REUSABLE ADA CODE CONSTRUCTS

The coding of adaptable Ada source code can be done with two main perspectives on specifying what is to be adapted. The adaptation implementation specified here is performed on the Ada text file types and subtypes in the IM.

The specific adaptations done are performed by a default tool or a user specified tool. It is possible to embed other adaptation tool specifications in the retrieve file. The retrieve file is equivalent to the RSP's Adaptation Specification. It specifies what object is to be adapted and how.

There are currently two adaptation tools available. Inputs of interest for both adaptation tools are the files and options provided to control the adaptation tool execution. Both tools take one file specifying the substitutions to be performed and a second file containing a list of objects (Ada source code) the substitutions are performed upon. Additionally, one of the adaptation tools allows options to further tailor the adaptations. There is an option directing the removal of commented out code sections that are not applicable to nor used in the adapted source code file. Refer to the Ada code fragment listed earlier in figure 5. The sections removed are those portions of the "--$IF THEN --$ELSE --$END IF" that evaluate to FALSE. This adaptation tool is a two pass parser, performing a substitution of IP parameters within these expressions. The expressions are then evaluated and the resultant file is again processed with the substitution file directives. The other adaptation tool is a single pass parser performing the substitutions one time through.

The use of either tool is dependent upon the creation of the original adaptable assets. The domain engineer creating an adaptable asset must know which adaptation tool will be used.

## 3.4. RULE-BASED ADAPTATION

Rule-based adaptation is based on a generator which evaluates the expressions (rules) in the AV's, DV's, and IP's. After the domain engineers have created the DM and all its supporting assets in a DE repository, this DM is available for application engineers to exercise a Decide Activity upon, i.e. answer questions from a DM node, thereby building a node in the AM. This AM will contain all answers given by the AE to questions in the DM. The AM with its answers in the form of AV's, in conjunction with the AV-DV mappings and DV-IP mappings, contains all information necessary to specify the adaptations of assets from the Reuse repository and place them into the AE repository.

The application engineer, during the question/answer session, need not completely answer all questions. Partial answering of a DM node's questions is permitted. The states of the answers are retained for further sessions. If the domain engineers created dependences to lower DM node questions, the lower node's questions cannot be accessed until all higher dependent questions have been answered. This dependency constraint is on the DV's in the DT's (not necessarily on all DT's in a design group).

### 3.4.1 DECIDE ACTIVITY (APPLICATION MODELING)

The AE begins with the Application Modeling Activity. The user is presented a list of available nodes in the DM upon which the decide can be performed. The AV's, DV's, and IP's, discussed earlier in section 2.3, are embedded in the DM. The application engineer proceeds in a top down fashion, answering questions at each node,thereby building a tailored DM, i.e., the application model (AM), in the AE repository as a hierarchical structure logically similar to the DM. Each node in the AM will contain or have relations to the answers to the questions in the applicable DM node.

Selecting a node in the DM and performing the decide method will result in the rule-base code being generated for an inference engine used by the SEE. The rule-base is comprised of the questions, question-help, potential answers, and dependency constraints upon other DV's which restricts or allows other questions. Additionally, this rule-base will contain the specification of how to output the results of the user answering the questions contained in the design group (a node in the DM).

Exiting a decide session will initiate the update to the AM in the AE repository. The update includes the creation of an applicable node in the AM, containing the answers from the corresponding DM node, and tagging the just completed node as Decided or Redecided as the case may be.

Updates include the next available DM nodes for Decide. This is directed by the dependencies resolved in the last set of questions answered. Answering certain questions (iterators or dependencies) allows the user to access other nodes in the DM. These nodes are only visible if the user answers questions in such a manner that all entry constraints to a DT are met. The answering of questions can also specify iterators which in turn specify the name of lower AM nodes.

The saved answers become a part of the applicable AM node. When later Decides are performed, all previous answers are gathered and provided to the inference engine. This allows the DE to specify dependency across various DM nodes that will not be apparent based on the nodes location in the DM hierarchy.

When the AE has sufficiently answered questions from the DM and has created an AM, the creation of retrieval specifications follows in the Application Production Activity.

### 3.4.2. APPLICATION PRODUCTION ACTIVITY

The process continues with the creation of retrieve files based on the results of the application modeling (section 3.4.1). The retrieve file is the mechanism by which the SEE specifies which objects in the Reuse repository are to be retrieved into the AE repository. It additionally can specify which objects are adapted and how they are adapted. After the retrieve file has been built, the Application Production activity uses copy and retrieve methods to produce an application in the application engineering repository.

### 3.4.3. COPY METHOD SUPPORT OF RETRIEVALS

The copy method refinement has been added to most types of interest in the IM. In the core COTS product, the methods for each type use a message dispatching mechanism and a structured list to contain details specific to the method. Each types' method can accept a message which invokes a specific method. The copy method is a Boeing extension to the repository.

The intent of the copy method is to ultimately create an object of a given type at the destination with all properties equal to an equivalent property value or state from the source object. The notion of an equivalent property has different meaning based on the copy refinement at a particular type.

The copy methods attempt to model the property refinement or property definitions used by the "new" method provided by the COTS tool. For named elements, a name property is provided. If the object has a description property, it is copied. Lower in the IM hierarchy at version, the location for creating the versioned object is provided. Lower yet at binary, three properties (storeType and either importedFrom or filePath) are provided. Whatever property the "new" method manipulates, the copy method at the same level in the type hierarchy also manipulates.

There is one major feature to the copy method refinements that is necessary to support the retrieval specification. The retrieve method which performs the actions specified in this specification (i.e. contained in a retrieve file), will pre-build a partial list based on the directives found the retrieve file. The copy methods must check for the existence of a refinement prior to the type unique refinement.

### 3.4.4. RETRIEVE METHOD OVERVIEW

The retrieve method has been defined on the two types used in the AM, the type used for the node instances and a file type. The retrieve method requires a location to create objects and two optional arguments.

The copy uses information derived from the location specified for the new objects and from the objects being copied. This information is used to locate the meta data type hierarchy for both the user's AE repository and the Reuse repository. These information must be kept separate even though the method definitions from each repository point to the same shareable images. The location to create the new object, an attach point, is used as the "destination" repository. The object receiving the copy message is assumed to be from the "source" repository. This object will be re-created in the "destination" repository using the applicable meta data types from the "destination" repository. This permits the copy to function between repositories or in the same repository.

The first optional argument, an enumerated selection, is used to determine if all copies specified in the retrieve file are performed and/or filtered to restrict the types copied, and also whether copied "as is" or "adapted". The second optional argument may be a list of types to be retrieved.

The selection option argument takes precedence over the filter type list. By default, the retrieve method will attempt to retrieve every COPY directive found in a retrieve file. If the filtering argument is present, it is used to compare against the type of the COPY directives found in the retrieve file. If the filter type argument is not present but the selection argument directs filtering, a Motif dialog box prompts the user for one of 4 categories of types; Requirements, Design Parts, Source Code , and Work Packages. These four categories map to a specific set of types.

There is a notion of scoping or nesting in the retrieve file BNF. BEGIN starts a scope, END closes it. The BEGIN/ENDs can be nested. Each COPY specified in the retrieve file is processed in a top down order. The COPY directive only states that an object is to be recreated in the destination repository. Whether or not adaptations are performed depends upon the ADAPT and SETPROP directives specified within the BEGIN/END scope containing the COPY and then higher scoped BEGIN/ENDs.

The retrieve method will initially create the lists used to specify what is created (type, name, various properties, etc.). It will place on the list any SETPROP directives specified in the retrieve file for each applicable COPY directive. This list is used in the "copy" message sent by the retrieve method to the COPY specified object found in the "source" repository. This is where the copy method takes over and finishes filling in the necessary entries for a "new" sent to the user's "destination" repository.

Currently, the adaptations by the TOOL directive applies only to the Ada file types. An internal parser is used to adapt all other files and description lists. The adaptation of descriptions is done internally for speed. The non-Ada file adaptation is simple substitution, line by line from the source file to the adapted file.

## 3.4.5. ADAPTATION TOOL SPECIFICS

The retrieve file BNF has a TOOL directive to accommodate adaptations of Ada files. The image specified immediately following the TOOL directive will be used for the adaptation of the instances specified for all applicable lower BEGIN/END scoped COPY directives. A sample retrieve file is shown in figure 6.

```
BEGIN
  ADAPT  from_3 to_3
  BEGIN
    TOOL  "pathname to image" -
    ADAPT from_1  to_1
    ADAPT from_2  "to string 2"
    COPY  type_1  name_1  name_
  END
  ATTACH   type_1  name_2
  COPY    type_2  name_3  name_
END
```

*Figure 6: Retrieve File*

The TOOL invocation uses any specified options and three files; a file with search and replace specifications, the Ada file, and a report file. The option shown above, -r, is the most pertinent here. It specifies the removal of unused --$IF THEN --$ELSE --$END IF clauses after the expressions are evaluated. The default is to leave the clauses, that do not evaluate to true, in the source code as comment lines.

This TOOL directive will override the default within a given BEGIN/END. All higher BEGIN/ENDs that have adaptations for an object will be gathered into a substitution file for input to the tool. One and only one TOOL directive is permitted within a BEGIN/END.

Various formats for specifying the search and replacement strings are supported. The format for the substitution file is line oriented, first token is searched for and replaced with the second token. Figure 7 contains an example substitution file.

```
$from_1$  to_1
$from_2$  "to string :
$to_$     to_3
```

*Figure 7: Substitution File*

This file will contain an ordered list of adapt specifications starting with the lowest nested BEGIN/END and proceeding up to any higher BEGIN/ENDs. Adapts within a specific BEGIN/END are listed in a top down order.

If used, the default SEE adaptation tool will apply all adaptations to the file's specially delineated lines (--$IF, --$ELSE, --$END IF, etc.). These expressions are then evaluated and the resultant file has the adaptations applied again. Prior to returning the file to the retrieve method, a commented out section will be prepended to the file. This section will contain the TOOL name with any command line options used to process the file followed by the substitution file (a list of all adaptations attempted). A sample of this header information in an adapted Ada component is shown in figure 8.

```
--
-- Adaptation done on yymmdd-hh:mm
-- Adaptation tool: "full path to ima
--
-- Search and replace list:
-- $from_1$ $to_1$
-- $from_2$ "to string 2"
-- $to_1$   to_3
--
-- Adapted results follows:
--
.... <adapted code>
```

*Figure 8: Adapted Component Header*

## 4.0. CONCLUSIONS

Effective automated support for product-line development can be provided with a combination of COTS tools and the Boeing/STARS developed software. Defining a reuse strategy is an essential step before an organization can achieve significant pay back from automation. Our automation strategic decision to use RSP was supported well by the adoption of the Boeing SEE together with development of generation technologies to support leveraged (versus opportunistic) reuse in AE processes. The technologies we developed are being demonstrated for the Air Vehicle Training Systems (AVTS) domain, but can be applied to virtually any domain.

The overall tailored RSP process will (as in the case of the Navy demonstration program) have characteristics of independence and specificity with respect to both the domain and the organization. RSP (both as defined by the SPC and as tailored for the demonstration project) can be applied to virtually any domain, and is thus domain-independent. However, since part of the DE task identified in RSP is the definition of the AE processes to be used by the given organization to build an instance of the product-line family, RSP is by definition always a tailored process. This tailored process will be specific at least to the organization and likely to the domain as well. This inherent recognition by RSP of the necessity for reuse to be "context sensitive" with respect to the organization and the domain is one of the strengths of RSP.

The reuse automation technologies that we developed were based on supporting the tailored RSP process. While Ada generics and object oriented principles supported in Ada 95 provide some support for reuse, they don't provide the generation capabilities and the reuse IM that our process dictated. Generation technologies that are coupled with rule-based dialogs can enable an organization to make use of reusable Ada assets without requiring the application engineer to understand all the details in the resulting Ada system. However, an application engineer may need to modify the generated system and will certainly need to test this generated system, therefore the generation technology must support generation of support documentation and test information. Our integration of reuse technologies with an object oriented repository has provided us with a capability to support not only generation of Ada code, but also the support materials needed by an application engineering project.

## 5.0. REFERENCES

[1] Software Technology for Adaptable, Reliable Systems (STARS), *Process Definition Guidelines*, Boeing STARS Technical Report CDRL 05150, Advanced Research Projects Agency (ARPA) STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, July 1993.

[2] Software Technology for Adaptable, Reliable Systems (STARS), *STARS Reuse Concepts Volume I - Conceptual Framework for Reuse Processes (CFRP)*, Paramax STARS Technical Report STARS-UC-05159/001/00, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, November 1992.

[3] Brad Cox, "Planning the Software Industrial Revolution," IEEE Software, 7(6):25-33, November 1990.

[4] Constance Palmer, "Identification and Tailoring of Reusable Software Components," Proceedings of the Fourth Annual Workshop on Software Reuse, Reston, VA, November 1991.

[5] Luqi and J. McDowell, "Software Reuse in Specification-Based Prototyping," Proceedings of the Fourth Annual Workshop on Software Reuse, Reston, VA, November 1991.

[6] Haikuan Li and Jan van Katwijk, "A Model for Reuse-in-the-Large," Proceedings of the Fourth Annual Workshop on Software Reuse, Reston, VA, November 1991.

[7] Karen Huff, Ronnie Thomson, and James W. Gish, "The Role of Understanding and Adaptation in Software Reuse Scenarios," Proceedings of the Fourth Annual Workshop on Software Reuse, Reston, VA, November 1991.

[8] Software Technology for Adaptable, Reliable Systems (STARS), *Reuse Library Process Model*, IBM STARS Technical Report CDRL 03041-001, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, July 1991.

[9] Software Technology for Adaptable, Reliable Systems (STARS), *The Reuse-Oriented Software Evolution (ROSE) Process Model*, Paramax STARS Technical Report STARS-UC- 05156/001/00, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, July 1993.

[10] Rub#n Prieto-Diaz, "Domain Analysis for Reusability," Proceedings COMPSAC '87, Tokyo, Japan, pp. 23-29, October 1987.

[11] Software Technology for Adaptable, Reliable Systems (STARS), *Reuse Library Framework (RLF)*, Paramax STARS Technical Report STARS-UC-05156/015/00, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, March 1993.

[12] Steven Wartik and Rub#n Prieto-Diaz, "Criteria for Comparing Domain Analysis Approaches," Proceedings of the Fourth Annual Workshop on Software Reuse, Reston, VA, November 1991.

[13] Software Productivity Consortium, *Domain Engineering Guidebook*, Technical Report SPC-92019-CMC, Software Productivity Consortium, Herndon, VA, December 1992.

[14] Software Technology for Adaptable, Reliable Systems (STARS), *Product-line Application Engineering Guidebook*, Boeing STARS Technical Report CDRL 05152, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, July 1993.

[15] NASA Software Technology Branch, "C Language Integrated Production System (CLIPS)" reference manuals, JSC-25012, September, 1991.

[16] Clinton Lalum, "Reusable Objects Access and Management System (ROAMS)," Abstracts of Posters Presented at the 15th International Conference on Software Engineering.

[17] ANSI, "Future Direction for Evolutions of IRDS Services," ANSI X3H4/92-161, September 1992.

[18] Software Technology for Adaptable, Reliable Systems (STARS), SEE Integration to Support Megaprogramming, Boeing STARS Technical Report CDRL 05104, STARS Technology Center, 801 N. Randolph St. Suite 400, Arlington VA 22203, June 1993.

[19] Margaret J. Davis and Harold G. Hawley, "Dialogue-Specified Reuse of Domain Engineering Work Products," Proceedings of the Eleventh Annual Washington Ada Symposium, McLean, VA, June 1994.

[20] Margaret J. Davis and Harold G. Hawley, "Reuse of Software Process and Product Through Knowledge-based Adaptation," Proceedings of the Third International Conference on Software Reuse, Rio de Janeiro, Brazil, November 1994.

[21] James R. Hamilton, "SEE Integration to Support Megaprogramming," Proceedings 1993 Software Engineering Environments Conference, Reading, United Kingdom, July 7-9, 1993.

## BIOGRAPHIES

Mr. James R. Hamilton has worked on the Boeing STARS project for over 5 years primarily working on software engineering environment (SEE) technologies. Mr. Hamilton lead the development of joint STARS documents in the area of SEE technologies and has lead the development of the Boeing STARS SEE. Prior to STARS work Mr Hamilton has worked in developing real-time Ada avionics systems for military applications. Mr. Hamilton holds a B.S. in Electronic Engineering from Oregon State University.

Mr. Harold G. Hawley has worked on the Boeing STARS project for over four years, initially organizing and managing the STARS "CASE Vendor's Workshop" in July, 1991, and subsequently as a member of the reuse team. Mr. Hawley was also Boeing's representative on the STARS program Reuse Joint Activity Group, which coordinated joint reuse activities by the three STARS prime contractors (Boeing, Loral, and Unisys) and played

an active role in reuse technology transition to the DoD, government contractors, and industry. Before joining the STARS project, Mr. Hawley worked with a small team which produced a prototype model-based reasoning tool to diagnose problems with electro-mechanical equipment. Mr. Hawley has been a lead software engineer on a number of aerospace and other real-time projects. Mr. Hawley holds a M.S. in Applied Mathematics from the University of Colorado and a M.S. in Computer Science from San Jose State University.

Mr. Clinton J. Lalum has worked on the Boeing STARS project since 1989. Initial work encompassed a comparative analysis of DCDS (Distributed Computing Design System) against earlier versions of the repository currently supporting the Boeing SEE. Subsequent work has been with the Boeing STARS reuse technology, designing and implementing the fundamental information model components and functionality of ROAMS (Reusable Object Assess and Management System). Before joining Boeing and the STARS project, Mr. Lalum worked on development of a PC based real-time spreadsheet, on enhancements and maintenance of embedded power monitoring systems, high speed flight testing data acquisition systems, and missile hardness surveillance systems. Mr. Lalum holds a B.A. in Computer Science and a minor in mathematics from the University of Montana.

Messrs. Hamilton, Hawley, and Lalum can be contacted at Boeing Defense & Space Group, P. O. Box 3999 - M/S 87-37, Seattle, WA 98027 (U.S.A). Their respective e-mail addresses are hamilton@plato.ds.boeing.com, hawley@plato.ds.boeing.com, and lalum@plato.ds.boeing.com.