An Approach For A Configurable and Accessible Environment Model

Lynn D. Stuckey Jr. Gary M. Kamsickas William V. Tucker
Boeing Defense & Space Group
Modeling & Simulation Technology
Huntsville, AL.

ABSTRACT

The creation and development of a synthetic environment for a simulation exercise can be a laborious, time consuming, and sometimes undefined process. This is especially disheartening because it involves re-inventing the wheel. A synthetic environment is common to most simulations, only the fidelity, extent, and interoperability of the environment varies. There are a multitude of decisions to be made and requirements to consider, many of them reaching beyond the intended goal of simulating an entity or exercise to collect valuable data. These decisions may involve large quantities of information pertaining to the integration of environmental models, simulation entities of differing fidelities and architectures, selection of models appropriate for the simulation, network communication architecture interfaces and data transmission mediums. There is currently no central repository for environmental or entity models or descriptive information about such models. The developer is faced with hard decisions that are typically out of their scope of expertise and redevelopment of a new environment to refine decisions can be costly and time consuming. There is currently no effective way to create a distributed synthetic environment, in a quick and adaptive manner.

This paper presents an approach to rapidly developing a synthetic environment using defined processes, architectures, and repositories of environmental software and data. The synthetic environment of interest in this paper is a family of models and related knowledge that provides for the simulation of the tactical and natural environments within which a simulation entity operates. The synthetic environment is not simply an exercise or scenario manager. The approach presented in this paper focuses on efficient information management with the goal of significantly reducing the time and effort required to create a synthetic environment and perform a simulation exercise. The approach involves architecture based selection and adaptation of environmental models, accessed through the world wide web and assembled by a knowledge based system. This approach allows the user to focus on the simulation at hand and not on the repetitive task of environment development.

BIOGRAPHICAL SKETCHES

Lynn D. Stuckey, Jr. is a software systems engineer with Modeling & Simulation Technology within the Boeing Defense & Space Group. Mr. Stuckey has been active in both applied development on several simulation projects as well as research into software development formalisms including process definition, methodology development, architecture definition, and cross-coupling of technologies such as knowledge-based systems, visualization, and simulation. Most recently, he has been involved in research and development activities dealing with leveraged software reuse through megaprogramming / product-line domain engineering. Mr. Stuckey holds a Bachelor of Science degree in Electrical Engineering and a Master of Systems Engineering from the University of Alabama, in Huntsville. He can be reached at lynn.stuckey@boeing.com

Gary M. Kamsickas is a Senior Principle Engineer with the Modeling & Simulation Technology organization of the Boeing Defense and Space Group in Huntsville, Alabama. He has been responsible for software design, code, test , Systems Engineering and integration on several Boeing simulation projects. Recent projects include Software Technologies for Adaptable, Reliable Systems (STARS) and Modeling & Simulation research and development. Mr. Kamsickas holds a Bachelor of Science degree in Electrical Engineering from Michigan Technological University, Houghton, Michigan. He can be reached at gary.m.kamsickas@boeing.com

William V. Tucker is the Modeling & Simulation Technology manager for Boeing Defense & Space Group, Concepts and Analyses organization. He has managed various trainer programs, including US, UK, and RSAF E-3, KC-135 and the Modular Simulator Design Program. He is a member of the DIS++ transition team and former chair of the DIS exercise management and feedback subgroup. He holds a Bachelor of Science degree in Electrical Engineering from Wichita State University in Electrical Engineering. He can be reached at william.v.tucker@boeing.com

An Approach For A Configurable and Accessible Environment Model

Lynn D. Stuckey Jr. Gary M. Kamsickas William V. Tucker
Boeing Defense & Space Group
Modeling and Simulation Technology
Huntsville, AL

INTRODUCTION

A synthetic environment is common to most simulations. The fidelity of the environment can range from a simple static atmospheric world with no other entities to a high fidelity, interactive world with dynamic weather and thousands of complex entities. Selection of the right environment for a certain simulation exercise can be a very difficult task. The simulation user may wish to determine the effects of certain environmental characteristics on a simulation in several "What If" type scenarios. In this type of analysis the user may not know the all the development requirements for the synthetic environment. This could cause the user to over specify, just to be safe, or only specify what is known at the time and make changes as new requirements are identified. Either way, the creation of a synthetic environment for a specific simulation can be very labor intensive and costly.

A recent goal in the area of software engineering is to increase software productivity in the creation of customized software systems through the concepts of software reuse and elevating the level of programming. The approach is to avoid building software systems one line of code at a time. By automating rote coding tasks and programming in high-level domain abstractions this can be accomplished. Through past experience we have found that the use of "reuse libraries" is not the best approach to increasing productivity. Software library reuse, or black box reuse, works only in fairly simple systems where whole components may be reused without change. In more complex systems customization is almost always required. In real reuse situations it seems that every component requires some kind of modification. Typical reuse libraries become roach motels; you can check in but you can't check out. What is needed is a method to build domain specific intelligence into components that allows automated modification for use in a specific application. It would be nice to allow the user to interactively specify their synthetic environment requirements through a well defined process and rapidly be provided with the model that meets those requirements. The user would then have the capability to rapidly generate a multitude of environment instances to exactly fit their simulation needs.

This paper presents a mid-term look at an ongoing internal research and development effort. We will discuss our current system concepts and processes, current progress and future plans for this research.

ENVIRONMENT DESCRIPTION

A typical synthetic environment is shown in Figure 1. The synthetic environment contains a representation of the real world, including an atmosphere and entities that operate therein. From the point of view of a single entity or object in the environment, such as an aircraft, everything external to the entity is considered that entity's environment. The synthetic environment always contains a natural component and may contain a tactical component.

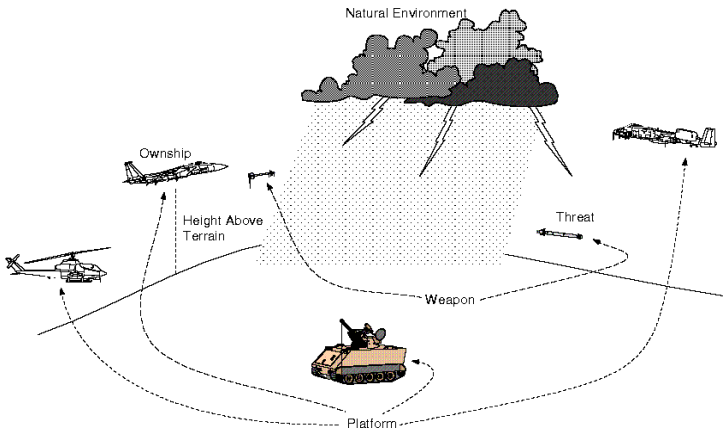

Figure 1. A God's Eye View of the Environment

Natural Environment

The natural environment contains naturally occurring (and static man made) elements. In the simplest case this may be a static atmosphere model, a flat earth, and no water. Higher fidelity representations of the natural environment include more features and more dynamics for the air, land, and sea. The air model might be extended to include three dimensional pressure wind and temperature variations, obscurants such as fog or clouds, and weather effects like rain, snow, or ice. Obvious land representation extensions include three dimensional terrain and dynamic terrain due to weather, traffic, or munitions. More complete ocean representations provide sea state, current, salinity profiles, temperature profiles, sea floor topology, and pressure effects. In the most complex case, extended features of the environment interact with each other causing such effects as terrain washout due to rain or beach erosion due to surf. The guiding principle is that the environment simulation should be precisely as complex as needed to accomplish the exercise objectives. Excessive complexity of the environment simulation provides unnecessary degrees of freedom in the post exercise analysis, unduly complicates the analyst's life and potentially reduces the validity of the results. It is easy to see that environment simulation varies widely in its use of potentially common elements.

Tactical Environment

The tactical environment consists of those environment elements that are primarily man-made and operate within the natural environment. Tactical elements may include other entities such as ships, aircraft, ground vehicles, missiles, bombs, or dismounted infantry. The tactical environment also includes the tactical effects of these entities operating within the environment. These effects include emissions from entities such as radar and electronic warfare signals, chaff, flares, biological and chemical dispersions, smoke clouds, aerosols, etc. Depending on the goals of the simulation exercise the tactical environment may be very complex, as in a sophisticated joint warfare simulation, or possibly even nonexistent, as in an operational flight trainer. The tactical environment may be represented locally for use by a single simulation entity or provided as a conglomeration of other distributed simulations such as is the case with Distributed Interactive Simulation (DIS) or High Level Architecture (HLA) based simulations. As with the natural environment, the tactical entities within the synthetic environment may interact resulting in a highly dynamic tactical environment.

Environmental Interactions

The complete representation of a synthetic environment is composed of the natural and tactical components and their interactions. Depending on the requirements for the simulation, these components and their resulting interactions can become quite complex. For example, tactical smoke clouds may drift and dissipate in the wind, the skin of a tank heated by the sun may cause a increase in its IR signature, electronic emissions can be affected or altered by the rain or salinity of the sea in which they are transmitted, or when a munition hits the terrain and detonates it may cause a transformation of the terrain. These environmental interactions can become increasingly complex and potentially an analysis exercise of its own accord. The modeling of these interactions can have a dramatic affect on the validity of a simulation exercise.

DESIGN APPROACH

Our approach to automatically generate synthetic environments is based on four elements; process, adaptability, architecture, and accessibility. Our process uses a proven two life cycle systematic, repeatable, reuse approach based on the use of software and data repositories. Adaptability is required to produce a synthetic environment that meets multiple user requirements. Our past experience has indicated that without a defined architecture it is difficult to obtain high level reuse and elevated levels of programming. The final element of our approach involves the automated selection and adaptation of environmental models through the use of the world wide web and a knowledge based system. This allows for a standard and distributed method of access for the user. The combination of these four elements allows the user to focus on using the simulation to perform the analysis instead of using valuable resources to generate synthetic environment models.

Process

Repeatable production of quality software begins with process. The Virginia Center of Excellence for Software Reuse and Technology Transfer created a process called the Reuse-Driven Software Process (RSP) for defining a domain, specifying the products available from the domain, and methods of adaptation to create instances of that domain. The goal is to implement designs that leverage whole and adapted components for reuse in new systems. RSP involves the definition, analysis, specification, and implementation of a domain encompassing a viable product family. Individual products are developed as instances of the domain, which reuse common

elements of the domain and adapt variable elements using a defined, repeatable process. RSP names the effort associated with creating the assets *domain engineering* and creating a specific product as *application engineering*. The central principle is that domain engineering creates the assets and processes that application engineering uses to create an instance of the domain (a specific product). Figure 2 illustrates this two life cycle approach.

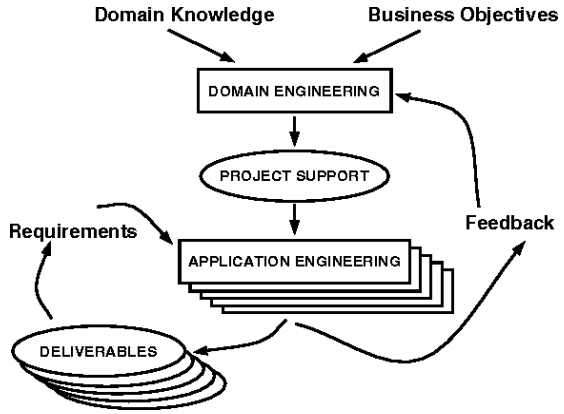

Figure 2. RSP is a Two Life Cycle Approach to Software Development

RSP was exercised and proven on the NAVY/STARS (Software Technology for Adaptable, Reliable Systems) demonstration project. This project was part of the US Department of Defense's STARS project. After several years of research and development, STARS adopted the concept of product-line development, with its two separate life cycles. The final task was to pair each of the prime contractors with one of the uniformed services in projects that would demonstrate the STARS technical thrusts in deployed systems.[Macala 96] The NAVY/STARS demonstration applied RSP to the Air Vehicle Training Systems domain, with a first product focus on the T-34C Flight Instrument Trainer(FIT). In November 1995, the project, having completed six iterations of the RSP, producing a flight dynamics simulation for the T-34C FIT. This simulation was proven to exhibit fidelity equal to or better than the existing fielded system. That is, it matched the performance of the real-world system at least as well as the existing simulation.[Macala 96] To demonstrate the power and flexibility of the domain-engineered assets, a second instance of the domain was created and tested within three hours. The domain supports many instances of flight dynamics simulation models. This demonstration proved the worth and potential of such an approach to simulation. Our approach applies this same process to the Environment domain.

Adaptability

Reusable simulation model libraries of let users view information, but not manipulate it. The Environment domain must support the creation of unique or custom instances of simulation models based on the immediate requirements of the application. To facilitate this requirement we are implementing three types of adaptability; Substitution, Pruning, and Modification. Substitution involves the selection of a component from a series of possible configurations. An example would be network protocols based on the user requirements for interoperability. Pruning is based on the notion that components may or may not exist in a given instance. An example would be a requirement for FAA wind profiles. If the requirement exists then the component is included, otherwise it is removed or pruned. The final type of adaptability deals with code modification. The notion is that certain variability is exhibited internal to a component. There are many times when an entire component does not need to change, only internal structures or algorithms. Examples of this type of adaptation are the setting of standard day constants, or the selection of dead reckoning algorithms.

Adaptability is provided automatically based on the results of the application engineering process. A decision vector is produced from the requirements provided by the user. These decisions are then interpreted through a metalanguage (the expert system) to adapt and retrieve the required components of the environment model. This adaptability is the culmination of the effort to produce an instance from the environment domain and relies heavily on the domain architecture. This technology was demonstrated and proven on the STARS demonstration project and is being leveraged into this project.

It is important to note that this adaptability produces different code than that of generic models or code generators. The basis for the model is constrained variability. The code is configured and optimized according to expert design, as defined by the user. The user gets a model that is efficient, of proper fidelity, and 'sized' to met the need. The user does not get a one-size-fits-all model that is forced to fit his/her situation. In the same light the user does not auto generate the model based on the graphical selection of static components. The adaptation process is more than just an automatic integration of predefined modules. Much hype surrounds the utilization of generic models and code generators. However, they have proven lacking in practical use. The problem has always been to individually constrain a solution that

meets the user's needs as if an expert had constructed the simulation from scratch against those requirements. Our approach is a step in that direction.

Architecture

We strongly believe that architecture is the enabling technology that brings the elements of process, domain, and automation together in a coherent approach to software development. Insightful selection and application of a software architecture is a key to the development of an environment domain. Indeed, product-line development rests on the assertion that it is possible to define an architecture that spans the domain and supports the creation of products for years to come. The architecture is the framework through which components, attributes, and inter-relationships within the system are expressed. An architecture consists of a partitioning strategy and a coordination strategy. The partitioning strategy leads to dividing the entire system into discrete, non-overlapping parts or components. The coordination strategy leads to explicitly defined interfaces between those parts. These two strategies provide an engineering approach to bridging the gap between the system as a whole and the design. The reach of such an architecture can extend from a single system to an entire family of systems. In the latter case, once the partitioning methods and coordination rules are determined, multiple products can be generated using the same methods and rules.

Maj. Robert Sonnemann, USAF stated several conclusions involving architecture in his dissertation "Exploratory Study of Software Reuse Success Factors". In particular, he stated three major points:

- "There is a STRONG relationship between reuse capability and having a common software architecture across a product line."
- "There is a STRONG relationship between reuse capability and having a software architecture which has proved useful in standardizing interfaces and data formats."
- "There is a relationship between reuse capability and having a software architecture which has greatly simplified re-hosting applications to different platforms." [Sonnemann 95]

For the development of the environment domain, we chose to utilize an established and proven architecture. This architecture is DARTS (Domain Architecture for Reuse in Training Systems).[Crispen 94] DARTS includes both a body of systems engineering work (common content) and a structural model (common component form). After this initial implementation, our plans are to look at an integration of HLA into this architecture as an interoperability scheme as well as a possible interconnection of internal objects. Figure 3 illustrates the DARTS structural model. This structural model is the basis for the implementation of our adaptable environment domain.

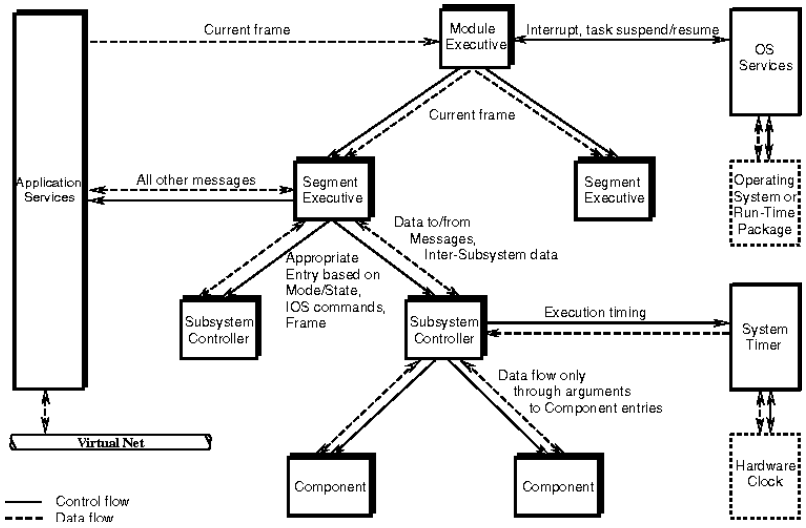

Figure 3. DARTS Structural Model

Accessibility

In RSP, application engineering creates products utilizing the assets provided by domain engineering. Accessibility is focused on the methods the application engineer uses to select, adapt, and deliver domain assets. The selection process is termed application modeling and is provided through an automated user

interface. The adaptation process, termed application production, is loosely based on an expert system. The delivery process is termed application delivery and is provided over the world wide web. The goal is to define and implement a process that provides the greatest possible accessibility to the potential users of the environment domain (Figure 4).

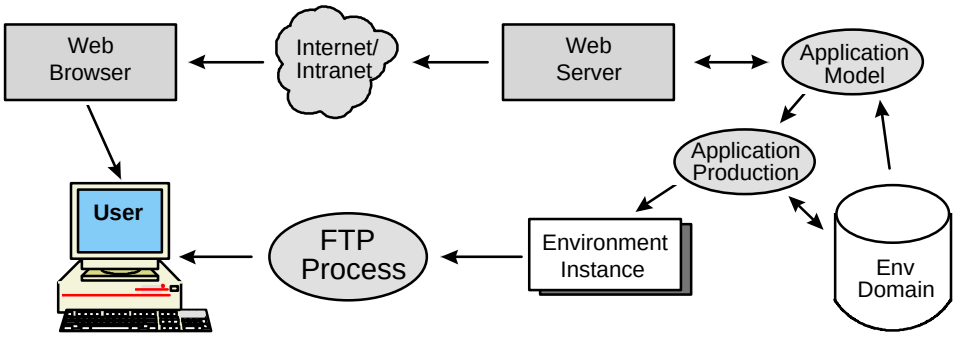

Figure 4. Accessibility to the Environment Domain

Application Delivery via the World Wide Web. The current feeling in industry is that if you aren't on the Web, you're behind. The next "mass medium" for business and education is the Internet. ("Couch potatoes" will be replaced by "mouse potatoes"). If you are looking for a mathematical routine, sort algorithm, general ledger package or database system that will save you from reinventing the wheel, you can now surf the Web for it. In the past, reusable assets have often been trapped within an organization because of cultural differences in organization and because of lack of connectivity between software libraries. Our accessibility approach, based on the Web, provides easy and convenient access to the environment domain.

The WWW could become the vehicle of choice for exchanging reusable software. One example of industry movement in this direction can be found in the Reuse Interoperability Group (RIG) in Arlington, Va. They have developed "bindings" that will let users in different organizations link software reuse libraries and exchange software via the World Wide Web. The bindings provide standards for conveying information about software assets to and from the Web. Software assets can include source-code routines, commercial packages, documentation, specifications or test suites.[Anthes 96] RIG efforts are providing a uniform method of access, enabling organizations to reuse assets which were otherwise unreachable due to lack of connectivity between libraries. "The WWW is becoming the vehicle of choice" for locating and exchanging reusable software, said Tim Niesen, a senior engineer at Raytheon Co.[Anthes 96]

The initial avenue of web access for the environment domain will be the internal Boeing web or intranet. Seeing the soaring popularity of the Internet and the global connectivity it provides, some companies use World Wide Web technology on corporate networks called intranets. An intranet is basically a private Internet that links a Web server and Web-browser clients over a local network. Intranet users are able to get out to the public Internet, but company firewalls keep outsiders from getting in. The ample availability of inexpensive browser software means that employees (whether they have PCs, Macs, UNIX, or a central kiosk) can access company information at a considerably lower cost that traditional groupware setups. Boeing is building upon previously developed electronic information delivery systems to use the Web as an integrating tool for both internally generated and externally published information across the company. Growth of the Boeing Web has made it clear that it is an effective communication tool for the entire company. As of January 1, 1996 over 250 separate sites are being indexed and classified under the subject codes on the library web server, and about 20,000 users are accessing the materials provided through the server. Transfer volumes are about 600 megabytes/day, and the library server is being hit 80,000 times a day on a consistent basis. Growth rates are approximately 4% per month, with no end in sight. The WebCrawler is handling over 2,000 queries per day.[Crandall 96]

Application Modeling via an User Interface. The utilization of the web is only the one step in the process of providing accessibility to our environment domain. The process of specifying a particular version of the domain assets for inclusion in the product is called application modeling. By exercising the application modeling process, an application engineer specifies the required product. As the application engineer makes decisions, these choices become the basis for defining and refining the form, fit, and function of the domain

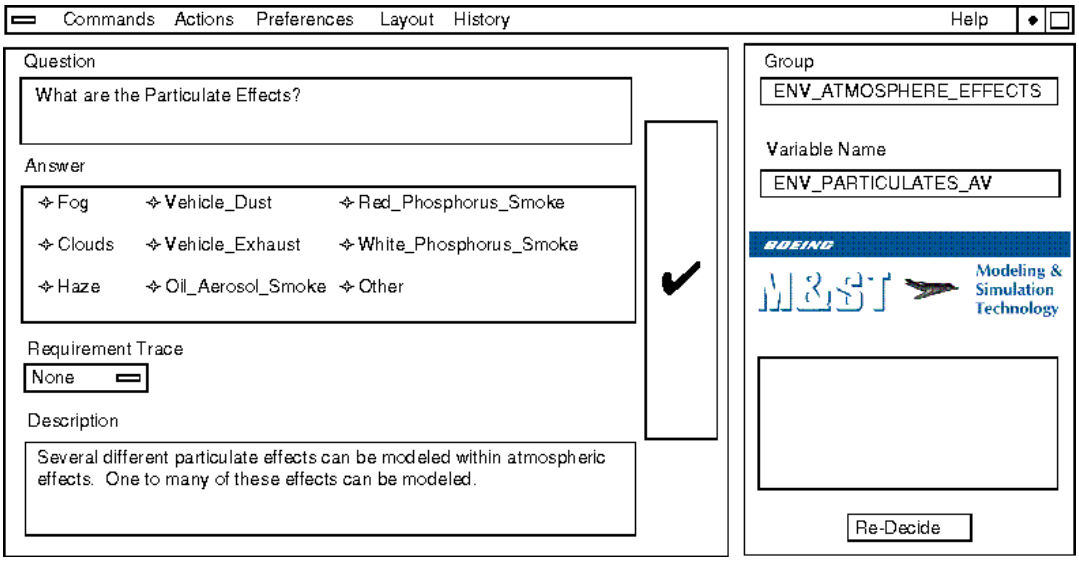

Figure 5. Example Application Modeling Scheme

assets that will be used to produce the final product. The number and kind of choices available to the application engineer are defined by a product of domain engineering in the decision model. An automated environment presents the application engineer with a screen for each choice(s) as Figure 5 illustrates. The automated environment tracks the decision made for use in the application production process. The STARS project demonstrated a similar product which is the basis for the environment domain user interface model.

Application Production via an Expert System. The result of the application modeling process is a record of decisions which form a specification of the desired product. The process of adapting domain assets to be that product is called application production. Application production creates a product in accordance with the specification created by application modeling. Application production of an instance of the environment domain is basically the execution of an expert system. An expert system is a computer program designed to model the problem-solving ability of a human expert. This system is composed of two principle parts; a knowledge base and an inference engine. The knowledge base contains highly specialized knowledge on the problem area as provided by the expert. It includes problem facts, rules, concepts, and relationships. The inference engine is the knowledge processor which is modeled after the expert's reasoning.

An expert system gains its power from the knowledge it contains. It is therefore important that every effort be made to assure that the knowledge that goes into the system effectively captures the expert's understanding of the problem. The task of interacting with the expert to acquire, organize, and study the problem's knowledge is formally called knowledge acquisition. The objective of knowledge acquisition is to compile a body of knowledge on the problem of interest that can then be encoded in the expert system. Whereas knowledge engineering is typically the bottleneck in the construction of an expert system, RSP's domain engineering provides a detailed process for this effort.

The focus of knowledge acquisition under the domain engineering process is the identification of commonality and variability across the product family (different instances of environment simulations), questions to ask that distinguish the variability, the presentation of the questions, and adaptation scheme/rules. The goal is to establish a standard expert system where a user is queried for certain facts and then rules are fired based on the facts asserted.

The three principle activities in Domain Engineering are domain definition, domain specification, and domain implementation. These activities correspond to creating a knowledge base and an inference engine. The domain definition activity characterizes how existing systems, systems begin developed in ongoing projects, and potential future systems are similar and how they differ. The domain specification precisely characterizes a product family for the domain and an process for constructing members of that family. The domain implementation activity implements the product design. This implementation is used by

application engineers to generate required work products for systems in the domain.

DESIGN ISSUES

In addition to the activities and goals associated with our design approach there are several other issues that should be considered. These design issues deal directly with several overall characteristics of the environment domain.

Legacy Model Interface

The creation of a synthetic environment repository that contains individual models of the environment components can be a very costly and time consuming task if created from a clean slate. However, there are a wealth of legacy systems that are capable of providing suitable models for the repository or as a minimum providing design guidance for the creation of new models. The use of legacy models can result in significant cost savings in the areas of software development, testing and integration. We have found that in the majority of cases legacy models cannot easily be "dropped into" the repository as is. Most legacy models were not designed with reuse or the concept of product-line software development in mind and some re-engineering of the legacy models may be required to account for software or hardware incompatibility. We must also consider the interconnection of other tools or projects that are currently used by the simulation community. A couple of good examples are the Modeling & Simulation Resource Repository (MSRR) and ModSAF. The MSRR can provide an interactive repository of simulation models. ModSAF is a simulation tool that could provide significant portions of a tactical environment.

Fidelity

The approach to generating the synthetic environment must consider that each environment instance may require different levels of fidelity based on the analysis requirements of the simulation exercise. In some cases a high fidelity, high resolution terrain model may be required to examine a specific mission characteristic. In other cases, a simple flat earth model may be all that is needed to conduct the simulation experiment. Synthetic environment fidelity requirements can be determined explicitly by simply prompting the user to indicate the fidelity desired or implicitly by letting the knowledge based system determine the fidelity level based on the user's description of the requirements of the simulation exercise. The goal is to avoid over specifying or under specifying the required environment. In most cases the user is simply going to know the characteristics of the environment required and the nature of the experiment. To ask the user to specify a fidelity level tends to be an ambiguous request requiring a quantification of what, for example, a high, medium or low fidelity level might entail. It would seem to be more efficient to embed the selection of the fidelity level within the knowledge based system. As the user specifies their synthetic environment requirements the resulting fidelity level is also determined thus eliminating the need to externally determine fidelity.

Scalability

Not every entity within the same simulation exercise will require the same environment or representation of the same environment. The high flying fighter aircraft is very interested in the atmospheric conditions in the direct vicinity of the aircraft and must be aware of the tactical environment including threats and emissions that directly effect the mission. However, this same aircraft is not concerned with the trafficability of the terrain or the temperature profile of the ocean beneath it. A ground vehicle is very interested in trafficability and a submarine highly aware of the ocean thermoclines yet neither is concerned with the aircraft’s atmospheric conditions. Each of these entities is operating within the same global environment but requires vastly different scales of resolution of data depending on the specific needs of the individual entity. The simulation exercise may require a very high fidelity synthetic environment, but it would be wasteful in terms of computational time and memory to provide each entity with exactly the same environment model when they require a small portion of the entire content. Our environment development approach must consider this scalability issue when identifying the environment requirements and generating an environment instance for a specific entity. If each entity simulation uses our approach to generate its individual environment instance we can ensure the environments are correlated for each entity. This improves the computational efficiency and promotes the concept of a leveled playing field among entities.

Commonality and Correlation

Much of the thrust of today's simulation exercises is focused on Distributed Interactive Simulation (DIS). In this type of simulation exercise there exists no central synthetic environment, each simulation entity must provide its own internal environment representation and maintain that representation

throughout the DIS exercise. A concern in this type of distributed exercise is the commonality and correlation of the synthetic environment modeled within each of the simulation devices. Using the right kind of common repository and associated processes to generate a synthetic environment instance for each distributed entity would improve the commonality and correlation of the synthetic environments among the entities. In addition, a significant amount of leveraged reuse would occur resulting in reduced development costs for all participants.

Development Cost

It is obvious that the initial development cost for this approach will be significantly greater than the cost to develop a single synthetic environment instance. This is typically the case with all reusable assets because additional considerations must be made to make the asset reusable across future applications. What is required is an estimate on the Return-on-Investment for the initial development cost. When will we see a financial benefit from the investment in the reusable asset? Our past experience in this process indicates that a Return-on-Investment will occur on about the third instance produced from the repository. Figure 6 provides probability cost curves for developing three systems using conventional and product-line development methods. This cost analysis was based on actual costs from a product-line demonstration project and Cocomo's software cost model estimates for producing future systems. [Macala 96]

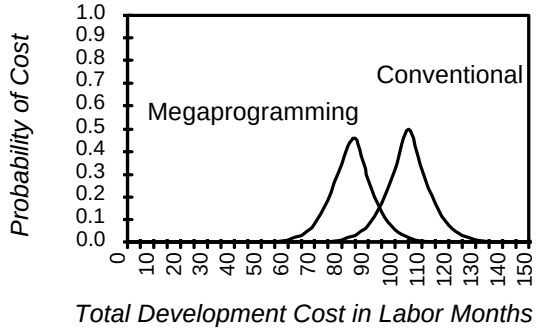

Figure 6. Return-on-Investment Comparison

However, we must also consider that the repository must be maintained to ensure that it remains current with the latest technologies and meets the yet to be determined requirements of future users. This domain engineering effort required to maintain the repository will be highly dependent on the design characteristics and completeness of the initial system.

STATUS OF IMPLEMENTATION

We have described our vision and process for implementing an Environment domain. We have completed several iterations of domain engineering, resulting in a sufficiently detailed collection of knowledge that describes the commonality and variability within different instances of environment simulation models. This activity has generated the following work products for the Environment domain:

- Domain Synopsis. An informal statement of the scope of the domain.
- Domain Glossary. Definitions of significant terminology used by experts in discussing needs and solutions in the domain.
- Domain Assumptions. A description of what is common, variable, and excluded among systems in the domain.
- Domain Status. An assessment of the current maturity and viability of the domain relative to is panned evolution.
- Legacy Products. A representative collection of work products from existing systems in the product line which may be a suitable source of information for developing the domain.
- Decision Model. A decision model identifies the application engineering requirements and engineering decision that determine how members of the product family can vary.
- Product Requirements. Product requirements determine the behavior and operational characteristics of problems solved by the domain.
- Process Requirements. Process requirements determine how application engineering is performed and which work products are produced as a result.
- Product Design. A product design determines the structure and composition of solutions provided by members of the product family.

Figure 7 illustrates the top level environment decision group. Figure 8 an example of explicit decision required in one of the decision groups. Our current effort is focused on domain implementation. We are implementing the set of adaptable components in parallel with the creation of the application model on the Boeing intranet

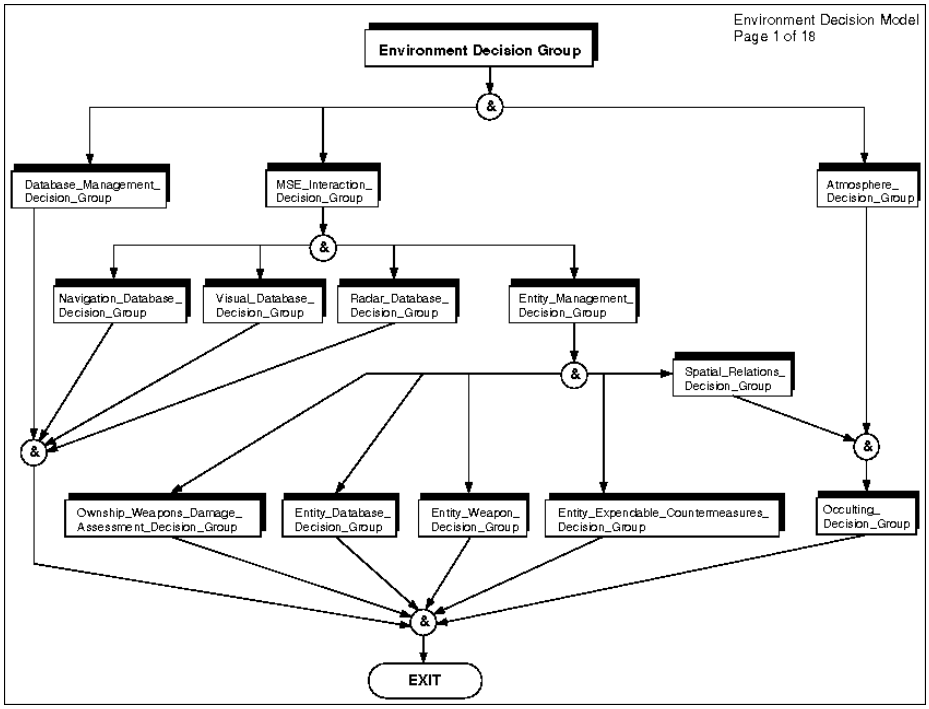

Figure 7. Decision Group Example

CONCLUSION

The feasibility of this approach to rapidly develop a synthetic environment using defined processes, a common architecture and repositories of environmental software and data is very high. We have successfully proven the concept of reuse using the two life cycle model of Domain and Application Engineering during the STARS demonstration project. The processes used to create and maintain the repository and associated assets have been exercised and improved upon through several iterations of the processes. We have seen the relative benefits of a common system architecture with a well defined communication infrastructure. Accessing software via a World Wide Web interface is currently being accomplished but not using the interactive approach described in this paper. We have developed a non-web based prototype Application Engineering interface into the repository with great success. Our intent is to transition the design concepts from this prototype to the web within the next year and populate a basic framework for a synthetic environment domain. We predict this approach will enable users produce high quality synthetic environment instances at a fraction of the traditional development cost and schedule.

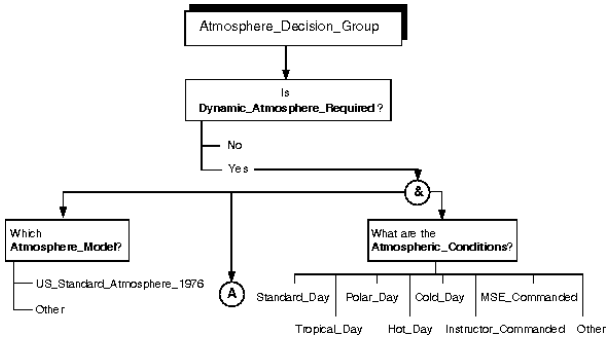

Figure 8. Decision Logic Example

REFERENCES

[Anthes 96] Anthes, Gary H. “Seekers of Reusable Code Can Now Pull it From the Web” *Computerworld*, May 13, 1996, p. 69.

[Crandall 96] Crandall, Mike and Mark Swenson, “Integrating Electronic Information through a Corporate Web”, *Computer Networks and ISDN Systems*, Volume 28, Issues 7-11, p. 1175.

[Crispen 94] Crispen, Robert G. and Stuckey, Lynn D. Jr., “Structural Model: Architecture for Software Designers”, Proceedings of the Tri-Ada 1994 Conference, Nov 6-11, 1994, pp. 272-281.

[Gross 95] Gross, David C., “Megaprogramming as a Reuse Strategy for Simulation”, Proceedings of the 1995 Southeastern Simulation Conference, Oct 23-25, 1995, pp. 244-249.

[Macala 96] Macala, Randall R., et al., “Managing Domain-Specific, Product-Line Development”, *IEEE Software*, May 1996, pp. 57-67

[Sonnemann 95] Sonnemann, Maj. Robert, USAF, "Exploratory Study of Software Reuse Success Factors" Ph.D. Dissertation, George Mason University, Fairfax, VA. 1995.

[VCOE 93] Reuse-Driven Software Process Guidebook, Tech Report SPC-92018-CMC, Version 2.00.03, Software Productivity Consortium, Herndon, VA. 1993.