Lisp and Portability:

The Process Modeling System

H. W. Egdorf
Los Alamos National Laboratory

A primary mission of the Technology Modeling and Assessment group (A-7) of the Analysis Division of Los Alamos National Laboratory is to support the Department of Energy in performing analysis of both existing and future facilities that comprise the Nuclear Weapons Complex. Many of the questions to be addressed in relation to this mission involve an examination of the flow of material through a processing facility and the transformations of the material as it moves through the facility by the use of a discrete-event simulation tool.

In support of these analysis tasks, a simulation tool kit has been developed that allows examination of issues related to the movement and transformation of material as it moves through a processing facility. This tool kit, The Process Modeling System, is currently the primary modeling tool used for examination of current and future DOE facilities. The flexibility of the system has led to its use in performing similar analysis on a number of non-DOE facilities under Technology Transfer initiatives.

The Process Modeling System is written in Common Lisp. The purpose of this paper is to describe the structure of the modeling tool kit and discuss the advantages of Common Lisp as its implementation language.

1.0 Structure of the Process Modeling System

The Process Modeling System is the implementation target of an object-oriented software engineering methodology that has been developed over the last eight years for support of development of discrete-event simulations. This methodology provides a structure by which an analyst progresses from a set of customer questions to a simulation tool to be used in answering those questions. Prior to use of the simulation tool kit, this methodology provides a design in terms of a set of objects, each object's attributes, and each object's capabilities. In addition, it will have been determined that the problem domain permits an analysis solution by use of a simulation tool examining constraints on material transport and transformation of material through a material processing facility.

It is the intent of the Process Modeling System that the set of objects identified by the software engineering methodology be implementable within the tool kit, or with simple exten-

sions to the tool kit when the problem domain is within the class of analysis problem described above.

1.1 Initial Structure - Control, World, and Artifactual Objects

The purpose of this section is to describe the types of objects (classes) that comprise the tool kit.

The discussion of the structure of the Process Modeling System will focus on those classes of objects that are visible to an analyst using the tool kit to construct an analysis tool. There exists one additional class of objects, Simulation Control Objects, whose function is to provide structure to the analysis tool. These Simulation Control Objects are not directly visible or accessible to the user of the tool kit. They may be accessed directly by programmers extending the tool kit, or indirectly as a side effect of interaction with other objects.

The primary example of a Simulation Control Object is the controller for the queue of events that comprise the discrete-event simulation structure. This event queue is not directly visible to the analyst constructing a simulation tool. This means that the event queue is not an explicit object with which the user can directly interact. Rather, the user (or a programmer extending the system) can make an opaque object called an event, and the action implied by the event will be performed automatically at the proper time in the simulation. Being a discrete-event simulation tool, these events are the agents that provide all of the activity in the system. However, the implementation of the discrete-event structure is not directly accessible to the user of the tool.

The set of objects visible to a user (either an analyst using the tool kit, or a programmer extending the tool kit) of the Process Modeling System provides an initial distinction between those objects that represent entities in the real-world situation being modeled and those objects that are artifacts of the fact that the system is a part of a computer analysis tool.

Those objects that represent entities in the real-world system under examination are called World Simulation Objects. Those that exist as artifacts of the computer simulation are Artifactual Simulation Objects.

Artifactual Simulation Objects in the tool kit include a class of extensible random deviates that provide for sequences of generated random variables and a class of displays that allows attachment of graphical interfaces written with different graphic interface systems but with a common protocol to the simulation model.

1.2 Primary Tool Kit - World Simulation Objects

The set of classes that comprise the largest part of the Process Modeling System are the World Simulation Objects. These objects represent those aspects of a real-world system that are commonly examined by a simulation tool in the Process Modeling System's problem domain of constraints on material transport and transformation of the form of the material during processing.

1.2.1 Material Representation - Containers

The Process Modeling System simulates material-processing activities within a processing facility. Several types of material containers are modeled. The proper type for a specific situation is chosen mainly by the level of detail of the model.

Material is implemented as a set of material-name/amount pairs. The materials themselves do not exist as objects in the model. The purpose of the various material containers is to contain these material-name/amount pair sets and provide various capabilities for their movement and handling.

Parts are the most detailed material container. Parts are both the container of material in preparation for a processing step (a material transformation) and also for transport from place to place in the model. A part contains a set of material-name/amount pairs and a workorder that describes where the part will move and what processing will occur at each step. Parts are used in the model when pure discrete processing of individual pieces of material is desired. Even when material is processed as a volume in a batch-wise manner, parts are used where the level of detail recognizes individual containers for the batches.

Where the level of detail of the analysis does not require explicit individual identity of Parts, a batch-processed aggregate container is provided. These Bulk Containers are similar to parts in that they contain a set of material-name/amount pairs. Bulk Containers are different in that they do not move from place to place and there is thus no associated workorder, and in that they allow the contained material to be processed in batches rather than as discrete whole quantities. A separate transport mechanism, streams, is used as the transport mechanism for these batches of material. These streams also provide a continuous processing capability where such is appropriate. Streams are provided associated with processing equipment described farther on.

For simple material representation where it is no longer necessary to view even aggregate containers, Material Resources can be used to provide a source of material that comes from a common inventory of similar materials. A conceptual model for these resources is a chemical stockroom or supply house that can provide amounts of various materials to be added to the system at a processing step. These materials can be limited in amount and thus constrain processing in the model, and they can be resupplied by processing in the model.

1.2.2 Material Transformation - Workcenters

The Process Modeling System simulates the processing of material as a transformation from one type to another. This transformation is provided by a class of equipment called Workcenters.

Workcenters are described by a set of process step descriptions. Each process step description specifies either a set of input parts, a set of bulk containers, or a single continuous stream. Each process step description then describes sets of material and other resources necessary for processing that can also constrain production. Finally, each process step

description describes the outputs to be produced. The processing described takes the form of a material balance where all material in the inputs is distributed to the outputs in such a way that all material added is accounted for in the output.

Each piece of equipment represented by an instance of class Workcenters can have several different behaviors depending on the types of input. Each Workcenter can also have stochastic failure and breakdown modes of operation.

1.2.3 Material Transport - Locations

The Process Modeling System simulates the transport of material from place to place in a processing facility. The class of objects that provides for constraints on this movement are Locations.

Each Location in the model can contain a set of parts, a set of bulk containers, and a set of workcenters. Subclasses of locations provide constraints on the movement of material into their instances. Existing constraints in the tool kit include limitations on the number of parts that may be in a location at any one time, area constraints limiting the floor space of the parts in a location, and material constraints limiting the amount of various materials in the location at any one time. A simple protocol allows for the creation of additional constraints as needed.

The operation of the model revolves around locations moving parts from place to place, and workcenters choosing material from parts, bulk containers, and streams, and producing outputs in the form of parts, material added to bulk containers, streams to other locations, and resource resupply.

1.2.4 Decision Making - Foremen

Each simulation tool constructed with the Process Modeling System generally requires some custom decision-making capability. These decision makers are modeled with the class Foremen. The result of a foremen decision is usually either the attachment of a workorder to a part, directing it through a set of processing steps, or the change of state of a workcenter (in effect, turning the equipment on or off.)

1.2.5 Data Examination - Auditors and Accountants

The Process Modeling System can provide large amounts of data while running. However, if nothing looks at those data, it is of little value. The conceptual model used for data examination was developed from the idea that the simulation tool is nothing more or less than a replacement for an experiment in the actual system under analysis where the cost of an actual experiment would be prohibitive. In an actual experiment, two sources of data gatherers, Accountants and Auditors, would exist.

Accountants are those procedures or individuals in the system that watch and record some behavior of an object in that system. Examples include parts entering or leaving a location, and a workcenter beginning or ending a processing step.

Periodically, an Auditor will arrive and balance the books of the accountant. The Auditor provides a time-stamped record of the model's operation at some periodic interval. It is these auditor records that generally provide the information with which the original analysis questions are addressed.

2.0 Why Lisp?

The Process Modeling System is, first and foremost, a specification of a software system. This system can be implemented in any reasonably object-orientated language. It is important not to lose sight of the fact that the choice of a language is relatively unimportant in a software engineering effort compared with the proper use of an object-oriented design and analysis methodology.

Given that the choice of language is not the major factor in the success of a project, it is still important to choose the best tools available for implementing a design. Thus, it is appropriate to examine the requirements of the design of the Process Modeling System that lead to the choice of Common Lisp as the implementation language of choice.

While several reasons can be given for the use of Common Lisp, including rich development environments and language features such as garbage collection, three factors are dominant in the choice of Common Lisp for the implementation of this system.

2.1 Object-Oriented Capabilities

The software engineering methodology used for construction of these simulation tools leads to an object-oriented design. The language chosen for implementation should support the object-oriented paradigm presented by the methodology in order to facilitate validation, verification, maintenance, and extension of the completed tool.

The Common Lisp Object System (CLOS) provides one of the most complete and mature object-oriented programming systems available. It fits the structure presented by the design quite well.

2.2 Decision Modeling

The software engineering methodology used for construction of these simulation tools makes an early distinction between those activities that are physical and those activities that are cognitive. While it is possible to model many of the cognitive activities with normal program code, Common Lisp provides a strong base for the construction of tools that make representation of these cognitive activities much easier than simply expressing the rules as program code fragments embedded in the rest of the model structure. Examples of such systems include forward and backward chaining rule systems, and neural network systems.

The ability to explicitly model decision-making activities as well as physical activities depends on more than just the language used to model the decisions. The knowledge and

view of the world in which the activity is embedded is also a part of this modeling. The result of a physical activity depends upon the actual state of the world. A cognitive activity, such as a decision process, depends upon the perceptions of the entity making the decision. Common Lisp provides a particularly rich environment for capturing and representing this knowledge about the environment in which these activities take place, and for differentiating between the physical state of the simulation world and the perceptions of this world by decision makers in the model.

2.3 Portability

Simulation tools constructed using the Process Modeling System must run in a variety of environments. Small desktop machines are used to construct and test the scenario files that describe the physical system being modeled to the simulation tool. Large workstations are a common choice for larger production runs that might take several minutes to several hours. For the largest problems, mainframe and super-computer class machines are desired.

While the technical issues of knowledge representation and object-oriented implementation provide strong reasons for the choice of Common Lisp for the implementation of the Process Modeling System, it is the issue of portability that provides the strongest argument for the language's use. The ability to provide an analysis tool on a desktop commodity computer such as a low-end PC or a Macintosh is, in actuality, not a wise thing to do. The problems which require a tool of this type are large enough to require a machine the size of a very high-end personal computer in order to have reasonable run times. The advantage of being able to run on the machines typically found on a desktop is one of marketing. If the customer can run (however slowly) on their familiar personal machine, they are usually amenable to use of a more appropriate type of machine for actual production use. The slow runs provide an initial familiarity that allows the initial acceptance of the product.

Workstation-class machines have proven to perform acceptably for analysts currently using the system. Even on a workstation, typical problems can take several hours to run, and a mainframe-class machine is desirable. This run-time is not a Lisp-specific problem. While there are theoretical advantages in run-time to simpler languages like C or Ada, this disparity is not a significant issue for this class of problem.

Lisp provides the greatest application portability available today. The development of the Common Lisp standard by ANSI X3J13 has allowed all vendors to produce a very portable base language. This sort of portability is not so different from the portability of languages like C or FORTRAN. However, following the common aphorism that anyone wishing to implement a Lisp-like application in another language must first implement much of Lisp in that language, the place where portability usually breaks down is in the support libraries that provide commonly needed functionality. While languages such as C, Ada, and FORTRAN have made great strides in the extension of language standardization to library standardization, the definition of Common Lisp itself already contains most of those things regarded as libraries in other language environments.

A significant advantage to Lisp's portability is the existence of the Common Lisp Interface Manager (CLIM). CLIM provides essentially the same user interface across a very wide range of architectures. It provides this portability by addressing interface design at a somewhat different level than other systems, many of which are specific to one operating system or windowing system.

The Process Modeling System includes an architecture that allows the attachment of a user interface written for any available graphic user interface. For example, a simple Lispview interface is available on a Sun workstation running Lucid Common Lisp and X11. A CLIM interface is attached to the system using the same protocols, as can be an interface for the Macintosh toolbox or for Microsoft Windows in Lisp systems that support those graphic user interfaces. Currently, only CLIM is available across the range of platforms on which the Process Modeling System runs.

2.3.1 Problems with Portability

No system provides everything desired, and Common Lisp is no exception. Several small items cause problems moving from one system to another. It should be emphasized that these problems seem minor to those experienced in similar exercises in other languages.

- Pathnames are different among different systems. It is difficult to provide a really seamless interface to the host operating system without exposing these differences. Lisp logical pathnames provide one solution, but this is still in conflict with users of some specific system who wish to use native pathnames.
- Many systems (but not all) provide a command-line interface to applications where command-line parameters are available to the application. The bit of the interface that collects these command-line arguments into a canonical form is system dependent.
- The ANSI specification is still a moving target, albeit a slowly moving one at this time. Each vendor provides a somewhat different snapshot of the specification as it existed at the time that vendor shipped its Lisp system. As the Process Modeling System makes little use of features that are still evolving, the only real problem with this has been a need to take some care in the choice of what packages to import in order to use the proper system definitions. This is easily handled in the initial load file.

It is also important to note what has not been a problem.

- Representations of numbers, either integer or floating point, need not be parameterized by system. For example, the random number objects in the model depend on knowing the characteristics of floating point numbers. Lisp provides sufficient information about arithmetic in a portable form that parameterization by system type is not needed.
- Parameterization by system type is not required to sort out the plethora of library routines that vary between different versions of UNIX, or between UNIX and MS-DOS.

3.0 Conclusions

For a simulation model specified and designed with an object-oriented methodology that includes specific information about decision making and that requires a high degree of portability, Common Lisp is the best choice of an implementation language. No other language provides the range of implementations, mature object-oriented features, ease of portability of the model, and ease of portability of interface.