Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

AN ARCHITECTURE FOR A GENERIC DATA-DRIVEN MACHINE SHOP SIMULATOR

Charles McLean
Al Jones
Tina Lee
Frank Riddick

Manufacturing Systems Integration Division
National Institute of Standards and Technology
Gaithersburg, MD 20899-8260, U.S.A.

ABSTRACT

Standard interfaces could help reduce the costs associated with simulation model construction and data exchange between simulation and other software applications -- and thus make simulation technology more affordable and accessible to a wide range of potential industrial users. Currently, small machine shops do not typically use simulation technology because of various difficulties and obstacles associated with model development and data translation. This paper provides an overview of work currently underway at the National Institute of Standards and Technology (NIST) to develop a software architecture, standard data interfaces, and a prototype generic machine shop simulator that can be readily, reconfigured for use by a large number of small machine shops. It also reviews prior work in this area and describes future work.

1 INTRODUCTION

In most cases, the effort required to develop a meaningful simulation for a small machine shop exceeds the resources available. Small shops typically do not have staff with appropriate technical qualifications required to develop custom simulations of their operations. If in-house staff or external consultants are available, shop management is often unwilling to invest the time, effort, and funding required for simulation modeling activities. Part of the reason is that model development is often complex and costly. This is due in part to the fact that commercial simulation software packages do not provide turn-key, i.e., ready-to-use, models for simulating job shop operations.

Furthermore, simulators are not designed to use traditional shop data in its native format, so models and data import routines usually must be developed from scratch. If simulation software vendors were to try to develop generic job simulation models, they are still faced with the problem that there are no standard formats for much of the data required to run the models. Thus, if someone wanted to input a specific shop's data into one of these hypothetical simulators, custom data translators would still need to be developed at possibly considerable expense.

1.1 Project Overview

The objective of our project and the research described in this paper is to develop a software architecture, standard data interfaces, and a prototype generic machine shop simulator that can be readily reconfigured for use by a large number of small machine shops. The software architecture is being defined using the Unified Modeling Language (UML) and Microsoft Visio. The data interface specifications are being developed using the Extensible Markup Language (XML) and Microsoft XML Notepad. The prototype simulator is based on Microsoft Visual Basic extensions to a commercial simulation package, Rockwell Automation's Arena.

The project described in this paper is a part of the NIST Systems Integration for Manufacturing Applications (SIMA) Program and the Software Engineering Institute's (SEI) Technology Insertion, Demonstration, and Evaluation (TIDE) Program. The TIDE Program is sponsored by the Department of Defense and Software Engineering Institute. TIDE is currently engaged in a number of other projects with various small manufacturers in the Pittsburgh, Pennsylvania area. The technical work is being carried out as a collaboration between NIST, SEI, Carnegie Mellon University, Duquesne University, the iTAC Corporation, and the Kurt J. Lesker Company. It is expected that results from this project will be published at its conclusion as a tool kit for small businesses in CD-ROM format.

The Kurt J. Lesker Company (KJLC) is an international manufacturer and distributor of vacuum products and systems to the research and industrial vacuum

markets. KJLC manufactures complete, automatically controlled vacuum systems with special emphasis on custom-designed, thin film deposition systems for research in alloys, semiconductors, superconductors, optical and opto-electronics. A small machine shop is contained within the KJLC manufacturing facility. KJLC's machine shop operation has been used to help define the requirements for simulation modeling and data interface specification activities described in this paper. Their facility will also be used as a pilot site for testing and evaluation of the simulation models, neutral data interfaces, and other software developed under this TIDE project. For more information on KJLC, see www.lesker.com.

1.2 Related Work

The problems involved in developing and maintaining simulation models using commercial tools are well known. A number of approaches have been proposed to develop models that can be used to generate executable simulations, easily or automatically. Several are described in the following sections.

1.2.1 Graphical Approaches

There are many graphical approaches that produce network-type models that generate simulations. Three such approaches are discussed: IDEF3 (where IDEF is defined as the Integrated Computer Aided Manufacturing, or ICAM, DEFinition 3 method), Petri Nets, and Message-based Part State Graphs (MPSG).

1.2.1.1 IDEF3

The IDEF3 Process Description Capture Method provides a mechanism for collecting and documenting processes, see www.idef.com/idef3.html. IDEF3 models the behavioral aspects of an existing or proposed system by capturing process knowledge, which is structured within the context of a scenario. This knowledge includes, temporal information, precedence relationships, and causality associations connected with enterprise processes. The resulting IDEF3 descriptions provide a structured knowledge base for constructing analytical, design, and simulation models.

There are two IDEF3 description modes: process flow and object state transition. A process flow description captures knowledge of "how things work" in an organization, e.g., the description of what happens to a part as it flows through a sequence of manufacturing processes. The object state transition network description summarizes the allowable transitions an object may undergo throughout a particular process. Both the Process Flow Description and Object State Transition Description contain units of information that make up the system description. These model entities, as they are called, form the basic units of an IDEF3 description.

1.2.1.2 Petri Nets

A Petri net is a graphical and mathematical modeling tool consisting of places, transitions, and arcs that connect them see pdv.cs.tu-berlin.de/~azi/petri.html. Input arcs connect places with transitions, while output arcs start at a transition and end at a place. Places can contain tokens; the number (and type if the tokens are distinguishable) of tokens in each place constitute the current system. Transitions are active components, which can fire and change the state of the system. Transitions are only allowed to fire if they are enabled, which means that all the preconditions for the activity must be fulfilled. When the transition fires, it removes tokens from its input places and adds some at all of its output places. The number of tokens removed/added depends on the cardinality of each arc.

Petri nets are a promising tool for describing and studying systems that are characterized as being concurrent, asynchronous, distributed, parallel, non-deterministic, and/or stochastic. As a graphical tool, Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems. To study performance and dependability issues of systems it is necessary to include a timing concept into the model. There are several possibilities to do this for a Petri net; however, the most common way is to associate a firing delay with each transition. This delay specifies the time that the transition has to be enabled, before it can actually fire. If the delay is a random distribution function, the resulting net class is called stochastic Petri net.

1.2.1.3 Message-based Part State Graphs (MPSG)

A message-based part state graph (MPSG) is a formal model of the execution portion of shop floor controllers, see tamcam.tamu.edu/mpsg/WEBMPSG.htm. MPSGs were designed specifically to serve as the input to an automatic, code-generation system for shop floor controllers. These controllers can be combined into a variety of control architectures. The MPSG model represents the distributed shop floor controllers as communicating finite state machines. As such, an MPSG formally defines the protocol that the controller accepts. In order to have an operational controller, however, this execution module must be combined with a planner and scheduler which perform the decision-making functions.

MPSGs were originally used to model and coordinate activities among pieces of equipment on the shop floor. Son (2000) extended this notion to the shop level of the control system. He also developed a technique to automatically generate a shop simulation that could be used for planning and scheduling, as well as execution.

1.2.2 Simulation Generators

Mathewson (1984) was one of the first researchers to discuss the use of program generators for simulation models. He defined a program generator as a routine that accepts a simple descriptive input of the system to be modeled and produces a correct simulation model as output. The descriptive model was created using entity cycle diagrams. The building blocks for these diagrams include queues, activities, events, and flags. The actual generator, called DRAFT, contained an input controller to parse the diagrams, a model analyzer, translator, compiler, and executor. This approach works well for systems analysis -- it can easily give system parameters such as throughput and delay. However, it was not designed to provide information about specific entities that flow through that system.

A number of domain-specific generators have been developed in recent years. The one described in Aytung and Dogan (1998) is typical of many such generators. This particular one is for a dual-card Kanban system. The authors present a framework for defining the systems and a generator for producing the simulation. The building block for the framework is the work center definition together with a number of rules for implementing the various operating decisions. Two separate files are generated: a model file, which is based on the framework, and an experimental file, which contains the experimental condition under which the model is run.

1.2.3 Data Driven Approaches

Pidd (1992) provides guidelines for developing data-driven, generic simulations. Pidd defines generic to mean that the model is designed to apply to a "range of systems with structural similarities." He defines a model to be data-driven if "any instance of a system may be fully specified to the model without any need for programming." This broad definition would imply that most models developed using commercial tools are data-driven. Pidd provides an architecture for a data-driven simulator that relies heavily on various libraries and a model configurator. This model configurator corresponds to the various templates or graphical user interface provided by most commercial tools. He recommends a separate file to store the data that is used to drive the model.

While it is a step in the right direction, Pidd's approach does not go far enough. The user must still be quite familiar with a specific commercial tool to build a usable model. Researchers at NIST have been developing an approach, which is the focus of this paper, that extends Pidd's notion of library. An initial NIST approach -- see Son et al (2002) for some early results – attempted to build a formal representation for all data needed to run the simulation. This data could then be stored in a database completely separate from the simulation. There are two major advantages of this approach. First, it is easy to modify the data when changes in the systems occur. Second, it minimizes the interactions the user has with the underlying simulation language.

1.2.4 A New Hybrid Approach

Our new hybrid approach, as described in this paper, creates a fairly comprehensive shop data model and exchange file format for data-driven simulation. It also separates simulation functionality into modules in a new way.

The shop data model currently encompasses a significant portion of the data required to actually run a real machine shop, not just simulate its operation. Links are also provided in the data format to reference data maintained in files that use other standards. This approach, i.e., based on a unified data format, was taken to ensure data consistency between the real shop and the simulated shop. It eliminates the need to abstract or simplify real shop data for the purposes of creating a simulation – resulting in the elimination of a major step that is usually required in the simulation modeling process. Of course, our current data structure definition is much larger than the one developed by Son that was mentioned in the previous subsection.

Another major difference between our current approach and the past approaches of others is to isolate most of the complex shop data processing from the traditional discrete event simulation engine. The sequencing of a hierarchy of orders, jobs, and tasks is handled by a machine shop emulator software module. The emulator also handles resource allocation (equipment, employees, tools, fixtures, inventory) and other functions, see Section 3.3. The emulator is separate but linked to the traditional discrete event simulator. The discrete event processor is responsible for managing the flow of objects through generic event queues and state changes.

Conceptually the system may be thought of as two sets of linked Petri nets. Each Petri net is responsible for governing a portion of the simulation problem -- one set of nets for managing the discrete event logic, the other for managing the shop data processing logic. A reassignable set of generic, proxy work items and resources are manipulated by the discrete event simulator. Proxies are reassignable representations of shop emulator objects, i.e., actual work items and resources. Proxy work items flow

through and between generic queues and await timeouts (computed by the shop emulator). Proxy resources undergo generic state changes and await similar timeouts. Emulator shop objects are temporarily linked to proxy work items or resources, as appropriate. Proxy objects may be released when their processing by the discrete event simulator ends.

The creation of proxy work items and selection of their next queue is determined by the machine shop emulator. Two examples of generic queues are: 1) work items awaiting a shift change, or 2) work items awaiting a machine breakdown. Further discussion of the intricacies of this approach is beyond the scope of this paper.

2 REQUIREMENTS DEFINITION

Requirements for this TIDE Project were developed over a period of several months and specified using the Unified Modeling Language (UML). Conceptual models for major data structures were also defined using UML techniques. The most significant project requirement was to integrate and implement a manufacturing execution system, a production scheduling system, and a simulation system to satisfy the operational needs of the KJLC machine shop. It was expected that modifications to all three systems would be necessary.

2.1 Why Use UML?

A prerequisite for performing a thorough examination of the operations of a job shop is developing a model of the shop. The model is a description of the manufacturing related entities in the shop and the activities that are to be applied to those entities. The model should support the specification of the characteristics of the manufacturing entities, the relationships between the entities, and the production activities that manipulate the entities. It should support descriptions of the shop from different viewpoints. This quality is needed to ensure that information that is important to the shop floor manager can be specified while also having a place to specify information important to the machine operator and the maintenance engineer. Furthermore, it should be possible to specify different parts of the model at different levels of rigor, and to increase the thoroughness of parts of the model as new insights into the shop's operations are gained. Finally, other modeling approaches are tied to a specific methodology, and none of these methodologies were designed with any consideration to their appropriateness for describing job shop of manufacturing operations.

UML was chosen to create the model of the job shop. See Alhir (1998) for a technical presentation of UML, or Schmuller (2002) for a more tutorial approach. In addition to supporting the requirements specified in the previous paragraph, it has several other advantages. It is a recognized standard for structured and object-oriented modeling. Inexpensive tools are available to support creating the models. It is a visual language based on simple visual constructs such as boxes and lines. It supports the encoding of large amounts of information in each diagram. Moreover, while the different diagram types allow different aspects or viewpoints of a system to be modeled separately, it is straightforward to show the relationships between constructs in different diagrams.

2.2 Machine Shop UML Diagrams

Most of our work has focused on creating the following types of UML diagrams: use cases, static data structures, sequence diagrams, and state/activity diagrams.

Generic use case diagrams were created to identify the various actors and the roles that they had in a generic manufacturing facility, such as the one found at KJLC. In use case diagrams, a stick figure is used to represent an actor, an ellipse is used to identify a use case (a system function or capability), a box defines the overall system boundary, and connecting lines indicate communications link or interactions between actor and the use cases. Lower level use case diagrams were also created that decomposed the high level manufacturing facility use case. The other use cases were:

- manage work force
- manage customer orders
- design products
- estimate orders
- plan production
- acquire goods and services
- manufacture products
- manage inventory
- maintain facilities and equipment
- model and analyze operations
- define simulation study parameters.

The last use case represents a lower level of depth in the hierarchical decomposition. It was defined because of our interest and project objectives that are focused on simulation modeling.

Due to the limited space available in this paper, we have chosen not to include use case examples as figures. Since our focus is machine shop simulation, the last two use cases from above are briefly summarized below.

"Model and analyze operations" includes the following actors: simulation analyst, plant/shop management, shop supervisors, project managers, and production staff. Lower level use cases include:

- define modeling objectives
- develop models
- code simulations
- test code
- validate code and models
- define study parameters

- capture study data
- run simulations
- evaluate results.

The "Define simulation study parameters" was further broken down into the sub-cases that might typically be relevant to a machine shop operation, namely: Specify study parameters for:

- capital equipment
- scheduling
- inventory
- work force
- tooling
- layout
- process
- maintenance.

Obviously each of the other use cases could also be decomposed further, but that was beyond the scope of our project.

UML static structure diagrams were used to create a high level conceptual definition for many of the data types that would be needed to support the "Manufacturing facility" use case. The static structure diagrams identify each of the major data types, their attributes, enumeration of value constraints for data attributes, and the relationships between major types. Diagrams were created for parts and bill of materials, group technology codes, customer data, internal organizational structure, production management data (orders, jobs, tasks, and schedules), process plans, manufacturing resources, equipment setup matrices, employees, skill definitions, calendars and shift schedules, inventory, simulation metrics, and statistical distributions. Since these data types will be discussed in more detail in a later section of this paper, their UML representation will not be presented here.

3 INTERFACES AND ARCHITECTURE

After completing the initial set of UML use case and static structure data models, the focus of our effort shifted to the development of a skeletal XML file format for machine shop data. Parallel work has been underway to create software prototypes that validate the proposed architecture was indeed feasible. In this section, we discuss our rationale for using XML for the neutral file, the machine shop data file format, the architecture of the machine shop simulator, and prototyping activities to date.

3.1 Why Use XML?

UML is a good choice for creating a model of the shop, but it is only a modeling language and not an implementation language. When related collections of information needs to be exchanged between the software applications that are a part of the shop, a different approach needs to be taken. The Extensible Markup Language (XML) is used to define the exchange formats for the information that was modeled using UML.

XML is a standard supported by the World Wide Web (W3C) standards body, see `www.w3.org`. It supports the development of structured, hierarchical data entities that contain a high level of semantic content, that is both human and machine interpretable, see DuCharme (1999) or Goldfarb (2002). Inexpensive tools are available to support creating and manipulating XML data. It is usually straightforward to translate data modeled in UML into XML.

Also, there are several supporting standards from the W3C that make working with XML easier. These include Document Object Management (DOM) for manipulating XML documents, XML Schema for defining the format of XML documents, and Extensible Style-sheet Language (XSL) for translating XML documents to other formats. Finally, XML is a textual format, as opposed to a binary format that would require special application support. Being a textual format facilitates XML's use with the integration mechanisms commonly available in existing commercial off-the-shelf (COTS) software applications (file import/export, sockets, pipes, etc.).

A number of software tools are available at little or no cost for working with XML structures. XML data files may be opened and viewed using Microsoft Internet Explorer (IE). XML structures may be edited and populated with data using Microsoft XML Notepad. A free beta release of XML Notepad may be downloaded from the Microsoft website, `www.microsoft.com`.

3.2 Machine Shop Data Model in XML

Although a complete exposition of the Machine Shop Data Model is beyond the scope of this paper, the XML file structure will be briefly introduced. See Figure 1 for an illustration of the top level of the XML machine shop data file definition. The image was created as a screen capture using the IE browser. The plus (+) signs indicate where there are one or more levels of substructure that may be expanded by clicking on the item. The minus (-) signs indicate where a level of structure has been expanded and clicking will close the substructure. A listing of the fully expanded skeletal file structure at all levels is approximately 60 pages in length.

The major elements of the top level of the XML structure are briefly described below.

Revisions is found repeatedly at many levels of the data structure. It provides a mechanism for identifying versions of subsets of the data, revision dates, and the creator of the data.

Units of Measurement specifies the units used in the file for various quantities such as length, weight, currency, speed, etc.

```
- <shop-data type="" identifier="" number="">
    <name />
    <description />
  + <revisions>
  + <units-of-measurement type="" identifier="" number="">
  + <departmental-structure type="" identifier="" number="">
  - <calendars type="" identifier="" number="">
      <name />
      <description />
    + <revisions>
    + <calendar type="" identifier="" number="">
    </calendars>
  + <skill-definitions type="" identifier="" number="">
  + <operation-definitions type="" identifier="" number="">
  - <resources type="" identifier="" number="">
      <name />
      <description />
    + <revisions>
    + <stations type="" identifier="" number="">
    + <cranes type="" identifier="" number="">
    + <employees type="" identifier="" number="">
    + <tool-catalog type="" identifier="" number="">
    + <fixture-catalog type="" identifier="" number="">
    </resources>
  + <layout type="" identifier="" number="">
  + <parts>
  + <bill-of-materials-group type="" identifier="" number="">
  + <inventory type="" identifier="" number="">
  + <process-plans type="" identifier="" number="">
  - <work type="" identifier="" number="">
      <name />
      <description />
    + <revisions>
    + <orders>
    + <jobs>
    + <tasks>
    + <history>
    </work>
  + <purchase-orders type="" identifier="" number="">
  + <organization-directory type="" identifier="" number="">
  + <references type="" identifier="" number="">
  + <probability-distributions type="" identifier="" number="">
  </shop-data>
```

Figure 1: Top Levels Of Machine Shop Data in XML

Departmental Structure defines the departments within the organization, their relations to each other, the positions and employees in each department.

Calendars identifies the shift schedules that are in effect for a period of time, breaks, holidays.

Skill Definitions lists the skills that an employee may possess and the levels of proficiency associated with those skills. Skills are referenced in employee resource requirements contained in process plans.

Operation Definitions specifies the types of operations that may be performed at a particular station or group of stations in the shop.

Resources describes all the resources that may be assigned to tasks in the shop, their status, and scheduled assignments to work items. The resource types available in the machine shop environment include: stations and equipment, cranes, employees, tools and tool sets, fixtures and fixture sets. Standard setups are also defined.

Layout defines the location of reference points within the shop, area boundaries, paths, resource, and part objects. It contains reference pointers to external graphics files that use appropriate graphics standards to further define these elements.

Parts provides elements for part specifications, group technology codes, customers and suppliers; as well as links to bill of materials, process plans, drawings, part models, and other references.

Bill Of Materials Group cross-references the parts and quantities required in a hierarchical bill-of-materials.

Inventory identifies the instances and locations for part, materials, tool, and fixture inventory.

Process Plans defines the routing sheets, operation sheets, and equipment programs that are associated with production and support activities. Routing and operation sheets correspond to the job and task level in the work hierarchy. The plans define the steps, precedence constraints between steps, and resources associated with the production of parts and performance of support activities.

Work specifies the hierarchy of work items to be processed within the shop, i.e., orders, jobs, and tasks. Precedence constraints defined in process plans are mapped to associated work items. Scheduling data and resource assignments for each work item are maintained in the structure, as well as other data. Jobs and tasks are cross-referenced to each other as well as routing and operation sheets respectively.

Purchase Orders identifies the internal and external purchase orders that have been created to satisfy part inventory requirements.

Organization Directory is used to maintain organizational data and contact information on customers and suppliers. Part, order, and purchase order data is cross-referenced to organizations and contacts in this directory.

References identifies external digital files and paper documents that support and further define the data elements contained within the shop data structure.

Probability Distributions defines distributions that are used to vary processing times, breakdown and repair times, availability of resources, etc. Distributions may be referenced from elsewhere in the structure, e.g., equipment resources maintenance data.

3.3 Simulator Architecture

The architecture for the generic machine shop simulator is divided into the following component elements:

- Neutral shop data file
- XML data processor

- System supervisor and reporting
- Machine shop emulator
- Discrete event simulator
- User interface system.

Each of the major modules of the architecture are briefly described below.

Neutral Shop Data File - This interface file, introduced above, is key to understanding the entire concept of the generic machine shop simulator. The file provides a mechanism for configuring the shop model and sharing data between simulation and other shop software applications. In the TIDE Project, other software applications are primarily the manufacturing execution and scheduling systems, although integration with other applications is envisioned in the future. XML is used to encode the data in the file.

The file contains not only executable or computable data to be processed by the simulation, but also descriptive text that is intended only for human interpretation. It also contains a network of cross-reference links between the various types of data required to plan and manage operations within the shop. It supports references to other external computer files and/or paper documents that provide more appropriate mechanisms or standards for encoding or representing data, e.g., part drawings. Subsets of individual data types, i.e., substructures, may be created, stored, and/or exchanged using the file.

XML Data Processor – This module is a library of routines that handle the import and export of data in the prescribed XML format of the neutral shop data file. The primary function of the module is to read the neutral shop data file in XML and translate the data into and out of the internal Visual Basic object structure of the *Machine Shop Emulator* module. It is also responsible for creating XML output files. A set of library subroutines that provide some of the functionality of this module are available as downloads from Microsoft as part of their implementation of W3C DOM specification.

Simulation Supervisor – This module is responsible for configuring the *Machine Shop Emulator* at initialization from data contained within the *Neutral Shop Data File* and coordinating the execution of the various modules during a simulation run; and outputs simulation reports.

Machine Shop Emulator – This module is responsible for managing the emulation of the operation of the machine shop and manipulating required data. This includes, but is not limited to, managing:

- the production calendar and shift schedule
- the execution of orders, jobs, and tasks
- the enforcement of precedence constraints in process plan routing and operation sheets
- the sequencing of steps in the operation sheet associated with an individual task
- the availability of resources: stations, equipment, tools, fixtures, cranes, employees, and inventory
- the allocation of resources to tasks
- the selection of statistical distributions to be applied to various processes and events.

The *Emulator* module is being implemented in Microsoft Visual Basic and the Rockwell Automation Arena's SIMAN programming language.

Discrete Event Simulator – This module contains the commercial simulation software system, generic modeling elements (e.g., queues, resources, timers), and an interface to the *Machine Shop Emulator*. Among other functions, it is responsible for providing master clock functions, event queues and queue management, randomization functions based on embedded statistical distribution generators, and accumulating statistical data on resource state changes and work items.

The *Discrete Event Simulator* sequences arrays of generic resources and work items that act as proxies for the "real" shop data objects contained within the *Emulator*. The proxy objects are timed and sequenced through generic queues and state changes that represent key events and state changes in the real shop, as reflected in the *Emulator's* data structures. The sequencing process is driven by the data constraints on associated work-item and resource objects within the *Machine Shop Emulator.* This separation of responsibility between the *Discrete Event Simulator* and the *Machine Shop Emulator* allows the shop configuration to be changed within the *Emulator* without requiring custom, manual software modifications to the *Discrete Event Simulator* module.

Rockwell Automation's Arena is being used as the *Discrete Event Simulator*. For more on simulation modeling with Arena, see Kelton et al (2001).

User Interface System – This module provides capabilities for creating and modifying shop data files, managing the display screens for configuring the system and observing simulation runs, debugging, and displaying results. It also is responsible for the generation of custom reports of simulation results.

3.4 Prototype Development

Prototypes that have been built to date include:

Shop Data Editor – A graphical user interface was developed using various controls within Visual Basic™ to simplify data entry and population of the internal object structure and XML-based data exchange file. Development of the data editor is continuing to support the input of data for the entire machine shop data model.

Machine Shop Emulator – Various prototypes were created to validate that the architectural concepts could be implemented in Rockwell Automation's Arena. The prototypes were used to determine that resources could be dynamically-created based on externally-defined data,

custom state definitions could be applied to resources, resources could be pushed through state changes programmatically from an external module, statistics could be properly collected on generic dynamically-created resources, and routing of work items could be externally defined and controlled. Some of the logic required for the Machine Shop Emulator had been validated in a previous project using the Promodel simulator.

XML data processor – A prototype was developed to import and export scheduling data using Microsoft's XML DOM library. It also transferred data into internal Visual Basic object structures. The prototype verified the structure of software that processes the XML files.

4 CONCLUSIONS AND FUTURE WORK

Simulation software vendors are well aware of the fact that simulation technology is underutilized by manufacturing industry. An innovative architecture for a generic machine shop simulator was presented in this paper as one step towards solving this problem. New simulators, based upon this architecture, could simplify the modeling process and improve simulation accessibility for one industrial sector.

Our near term work is focused on constructing a prototype simulator based on this architecture using COTS software. We are also working with KJLC to populate the neutral data file with actual industrial data. Finally, we expect to integrate and test the prototype simulator and test data files with the other TIDE software applications.

A key factor in increasing the utility of the simulation is the establishment of standard data formats. This paper also presented a neutral data format for representing and exchanging machine shop data in XML. In the more distant future, our work will focus on expanding the data file format to incorporate additional data types, including a more sophisticated plant layout data model. We are working with the industrial team that developed the Simulation Data Exchange (SDX) plant layout format. We expect to adapt and integrate the SDX format with our shop data model. Our ultimate objective in this area is to promote the establishment of a standard data interface for manufacturing simulators based upon this work.

As the generic shop simulator becomes operational and standard simulation data formats are adopted, we see the possibility of new uses for simulation technology. One such use is the dynamic testing and evaluation of other manufacturing software applications. Currently, due to the lack of interface standards and testing systems, it is virtually impossible to adequately test new manufacturing software applications. Simulation technology, by enabling the implementation of virtual production facilities, can be an important new tool in the software testing arena.

REFERENCES

Alhir, S. 1998. *UML in a Nutshell*. Cambridge: O'Reilly & Associates.

Aytung, H. and Dogan, C. 1998. A Framework and a Simulation Generator for Kanban-controlled Manufacturing Systems, *Computers in Engineering*. 34(2): 237-350,

DuCharme, B. 1999. *XML: The Annotated Specification*. Upper Saddle River, New Jersey: Prentice Hall.

Goldfarb, C. 2002. *XML Handbook*. Upper Saddle River, New Jersey: Prentice Hall.

Kelton, D., Sadowski, R., and Sadowski, D. 2001. *Simulation With Arena*. New York: McGraw-Hill.

Mathewson, S. 1984. The Application of Program Generator Software and Its extensions to Discrete Event Simulation Modeling. *IIE Transactions*. 16(1): 3-18.

Pidd, M. 1992. Guidelines for the Design of Data Driven Generic Simulators for Specific Domains. *SIMULATION*. 59(4): 237-243.

Schmuller, J. 2002. *Sams Teach Yourself: UML in 24 Hours*. Indianapolis: Sams.

Son, Y. 2000. Simulation-based Shop Floor Control: Automatic Model Generation and Control Interface. Doctoral Dissertation, Department of Industrial and Manufacturing Engineering, Penn State University, University Park, Pennsylvania.

Son, Y., Jones, A., and Wysk, R. 2002. Component-based Simulation Modeling from Neutral Libraries. *Computers in Industry*. In review.

ACKNOWLEDGMENT

Mention of commercial products in this paper does not imply approval or endorsement of any commercial product by NIST. This project is funded [in part] by NIST's SIMA Program and the SEI TIDE Program. SIMA supports NIST projects applying information technologies and standards-based approaches to manufacturing software integration problems. The work described was funded by the United States Government and is not subject to copyright.

AUTHOR BIOGRAPHIES

CHARLES MCLEAN is a computer scientist and Program Manager of the Manufacturing Simulation and Visualization Program at NIST. He also leads the Manufacturing Simulation and Modeling Group. He has managed research programs in manufacturing simulation, engineering tool integration, product data standards, and manufacturing automation at NIST since 1982. He has authored more than 50 technical papers on topics in these areas. He is on the Executive Board of the Winter Simulation Conference and the Editorial Board of the

International Journal of Production, Planning, and Control. He is formerly the Vice Chairman of the International Federation on Information Processing (IFIP) Working Group on Production Management Systems (WG 5.7). He is also the NIST representative to the Department of Defense's Advanced Manufacturing Enterprise Subpanel. He holds a MS in Information Engineering from University of Illinois at Chicago and a BA from Cornell University. His e-mail address is mclean@cme.nist.gov.

ALBERT JONES is an operations research analyst and currently manages the Enterprise Integration Program at NIST. Prior to that, he led several projects to investigate the functional and integration requirements for the next-generation simulation tools. Dr. Jones has published numerous articles on control systems, scheduling, and simulation. Before coming to NIST, Dr. Jones taught for several years at John Hopkins University and Loyola College of Baltimore. He received his Bachelors Degree from Loyola College. He received a MS in Mathematics and PhD in Industrial Engineering from Purdue University. Dr. Jones is currently on the Executive Board of the Engineering School at Loyola of Baltimore. He has chaired or co-chaired several international conferences, and has served on several proposal evaluation panels for National Science Foundation (NSF), NIST, and Defense Advanced Research Projects Agency (DARPA). His email address is jonesa@cme.nist.gov.

TINA LEE is a computer scientist in the Manufacturing Simulation and Modeling Group at NIST. She joined NIST in 1986. Most recently, she has been working on the design and development of interface information models to support the Software Engineering Institute (SEI) Technology Insertion Demonstration and Evaluation (TIDE) project. Previously she worked at the Contel Federal Systems and at the Sperry Corporation. She received her BS in Mathematics from Providence College and MS in Applied Science from College of William and Mary. Her e-mail address is leet@cme.nist.gov.

FRANK RIDDICK is a computer scientist in the Manufacturing Simulation and Modeling Group at NIST. He has participated in research and authored several papers relating to manufacturing simulation integration and product data modeling. He is the NIST representative to the Distributed Simulation Special Interest Group within the Object Management Group (OMG) that is developing standard for distributed simulation. He holds a BS in Mathematics from Saint Augustine's College and MS in Mathematics from Purdue University. His e-mail address is riddick@cme.nist.gov.