Proceedings of the 2011 Winter Simulation Conference
S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.

MODEL BUILDING WITH CORE MANUFACTURING SIMULATION DATA

Jonathan Fournier

Connecticut Center for Advanced Technology
222 Pitkin Street
East Hartford, CT 06108, USA

ABSTRACT

The Core Manufacturing Simulation Data (CMSD) data interface standard created by NIST was developed to facilitate the exchange of manufacturing data between disparate manufacturing software applications including process planning systems and discrete event simulation tools. An Applications Programming Interface (API) was created for abstracting away the implementation details of reading and writing CMSD files to/from computer memory. This paper will detail the API structure and the structures of several translators which use the CMSD API. A simple case study will show the results of translating a CMSD file into several different simulation packages.

1 INTRODUCTION

Most commercially available Discrete Event Simulation (DES) packages provide some data- or control-interface for allowing users to quickly populate modeling elements. The speed of model building can be considered an important criteria in selecting a simulation package, as many simulation practitioners' time comes at a premium. Automation in model building can also help to reduce errors in duplicating data between a data source and the simulation package (Paprotny, Mackulak and Zhao 1999).

For a simulation practitioner, utilizing these automation mechanisms typically comes as a result of learning to manually build models in a particular package. The user gains an understanding of the modeling paradigm of the simulation package and how repetitive model building and duplicating data can get, and sees opportunities for automation.

Using these automation tools can be as easy as transforming a spreadsheet of input data to a different input format, specified by the package's documentation. It could get more complicated, requiring the user to write automation scripts in some programming language chosen by the simulation package developers. Users without programming experience can then be precluded from utilizing these tools, unless they first learn some generic programming techniques.

While these tools tend to work well for their designated purpose, there exists a large disconnect in how data gets used when building models in one simulation package versus another. This makes the sharing or reuse of simulation data cumbersome, often requiring that users re-enter the data or develop custom single-use solutions (Leong, Lee and Riddick 2006).

Once a model has been built in some simulation package, the data that defines the model is typically trapped in a format proprietary to the simulation package. Accessing that data then typically requires the development of custom software to interact with simulation package scripting interfaces, or custom parsers to read the package's model format (Wu 2005).

The Core Manufacturing Simulation Data (CMSD) was created to provide a standard format for holding the data a user would feed into a simulation model. Using the CMSD can alleviate the issue of manually reformatting data any time it is reused. The CMSD also provides an opportunity to move devel-

opment practices from one-off translation code to more general-purpose, off-the-shelf software tool development.

If a company uses multiple simulation packages for different projects, having CMSD translators for those packages would enable the users at that company to easily share data between projects. Also, any proprietary data extraction tools (i.e. from enterprise databases) they have developed could export to the CMSD format, rather than tailoring the export to one specific simulation package. Having these tools for flowing data between management information systems and simulation packages can even provide a mechanism for regenerating and re-running simulation models based on up-to-date information, presumably without a large amount of custom programming.

2 CORE MANUFACTURING SIMULATION DATA OVERVIEW

The Core Manufacturing Simulation Data (CMSD) effort was organized to address interoperability issues between manufacturing applications and simulation systems. The CMSD information model (CMSDIM) defines a data specification for the efficient exchange of manufacturing data in a simulation environment. The specification provides a neutral data format for integrating manufacturing software applications with simulation systems.

The CMSD has the following goals: (1) to enable data exchange between simulation systems, other software applications, and databases, (2) to support the construction of manufacturing simulators, (3) to support testing and evaluation of manufacturing software, and (4) to support manufacturing software application interoperability. The initial CMSD information model specification is presented in two different documents: (1) the information model defined using the Unified Modeling language (UML); and (2) the information model defined using the eXtensible Markup language (XML). The CMSD UML-based product development has been completed. CMSD became a standard product, SISO-STD-008-2010 (SISO 2011), of the Simulation Interoperability Standards Organization (SISO) in September 2010. Development of the CMSD XML-based product is planned.

The major data categories of manufacturing information included in the CMSDIM include: organization, calendar, schedule, work, process plan, operation and routing definition, part, bill-of-materials, inventory, people and skill definition, resource, setup definition, maintenance definition, distribution definition, revision, reference, and unit defaults.

3 CMSD APPLICATIONS PROGRAMMING INTERFACE

The CMSD specification provides a standard eXtensible Markup Language (XML) (Microsoft 2011) file format for representing a manufacturing system's data. Rather than write or re-implement code to generate or parse an XML document, a general-purpose CMSD Applications Programming Interface (API) was created. This API is compiled for Microsoft Windows as a Component Object Model (COM) library.

This environment was chosen as it is a mature, commonly used method for providing communication between computer programs in Microsoft Windows. In addition, advanced users of Microsoft Office technologies such as Excel or Visio are often familiar with Visual Basic for Applications (VBA), which uses COM technology for providing a programming interface to the Office products. It was assumed, then, that creating a reusable COM API for CMSD would provide simulation practitioners with programming experience in VBA with a low barrier of entry in creating their own CMSD translators. This paradigm for creating a CMSD API is consistent with guidelines provided by (Johansson et al. 2008).

3.1 Overall API Design

The CMSD API was implemented using a subset of the classes from the CMSD standard which were believed to provide a basic level of functionality for exchanging manufacturing data in a simulation environment. Further classes were added over time to enhance the functionality of the API, and in turn provide more data to translators being implemented.

The initial construction of the CMSD API provided the following classes: Part Type, Process, Resource, Process Plan, and Map. Each class implements methods for persisting and retrieving its properties to/from XML elements. The Map class acts as a container class for all the CMSD objects needed for a simulation model, and implements methods for persisting and populating CMSD objects to/from an XML file. Resource location is also supported (Riddick and Lee 2008)

Additional supporting classes have been added to the CMSD API following the same design pattern of providing methods for abstracting away the detailed implementations of working with XML files or elements. This leaves the developer of a translator free to focus simply on interacting with the CMSD objects

3.2 Extras (plugins, distributions, properties, scratch objects)

In order to provide some enhanced functionality, as well as making translators easier to write, several additional classes were designed and implemented within the CMSD API. These classes were designed after the experience of writing several basic CMSD translators.

For convenience, collection objects were added to most CMSD classes in order to maintain references between CMSD objects and objects in other software packages. Prior to adding these collections, a programmer was responsible for handling these references themselves, using whatever data structure they might like to use. These collections act somewhat like a scrap of paper for performing quick and dirty calculations; it just needs to be around long enough to get the information you need.

4 TRANSLATING BETWEEN SIMULATION WORLD VIEWS

The implementation of a simulation software package tends to follow some world view, or "framework for defining a system in sufficient detail that it can be executed to simulate the behavior of the system" (Pegden 2010). The CMSD standard was designed to package simulation data in a way such that the data can be used with any world view.

4.1 Process-based world view

The process modeling world view represents a system where entities move from process to process. Each process is composed of some sequence of actions, like requiring a resource, delaying for some time, and then releasing the resource. This is the world view adopted by the Arena simulation package.

4.2 Resource/Object-based world view

The resource modeling, or object-based, world view represents a system as a collection of physical objects located within a system, with entities moving from location to location for processing. When an entity arrives at a location object the object must decide what processing step to perform. Each process may require additional resources or entities, which may also impact its decision on what to process.

4.3 CMSD world view

The CMSD specification represents a system as lists of objects of various classes, with most objects referencing other object classes. For instance, the Process class holds a list of the Part Type objects that it requires for processing, as well as the products it creates. The Process class also holds a list of the resources required for processing. These resources can include physical locations (e.g. machines) and mobile resources (e.g. laborers or work fixtures). There is also a Process Plan class, which simply holds a list of Process objects in some sequence. This sequence can be used to describe the routing of a Part Type object. A Part Type object is used to describe some distinct entity in the system.

To translate a CMSD Map object to a process-based world view, the translator would have to look at each Process object and create that as a processing block, requiring and subsequently releasing the appro-

priate resources. The translator would then have to connect the different processes together based on the sequences provided by Process Plan objects.

In order to translate a CMSD Map object to a resource/object-based world view, the translator would have to first create all the fixed and mobile resources within the simulation modeling environment. Then, the translator would have to configure each location for processing entities (e.g. specifying additional resources, delay times). The translator would have to specify entity routing rules based on Process Plan objects.

5 SIMULATION MODEL BUILDER CONSTRUCTION

Several translators were created in order to test the robustness and applicability of the CMSD standard and CCAT's implementation of the CMSD API. Each translator was prototyped using VBA for ease of debugging, and later implemented using Visual Basic 6.0 as a COM Dynamic Link Library (DLL). These libraries could be reused easily for building Graphical User Interfaces or automated translation software.

5.1 CMSD to QUEST Translator

The QUeueing Event Simulation Tool (QUEST) (DELMIA 2011) provides an interactive 3D environment for building simulation models, as well as extensive scripting languages. The Batch Control Language (BCL) is similar to a shell scripting language and was designed to execute menu commands specified through a text input, bypassing the default interactive mouse-clicking interface. QUEST also provides an interpreted scripting language called the Simulation Control Language (SCL). The SCL language is used as the model run-time language for executing actions and logics within the different objects in the model. SCL can also be used for writing macro-level code to execute BCL commands.

The QUEST tool uses an object-based world view, meaning machines, queues, and laborers are located by the user within the 3D world. QUEST also provides a construct for specifying different Part classes. Part classes act as blueprints for generating part instances, which move around the modeled system for processing. The part class also holds an optional data structure for specifying the sequence of required processing for the part class. Each processing step is defined by a Process object. The Process object contains a list of the required parts, produced parts, and any resources required. Each Machine object holds a list of the Process objects it is able to handle. Through the run of a model, a machine will identify any Process object it can meet the requirements of, and executes that Process. The object model of QUEST closely matches the object model of the CMSD, making the creation of a CMSD to QUEST translator relatively easy, provided the CMSD API for handling CMSD data.

The CMSD to QUEST translator was designed to generate a BCL script that, when executed by QUEST, will populate the model with all the information from the CMSD Map object. The translator starts the script by creating all the locations and mobile resources specified in the CMSD Map's list of Resource objects. The translator then generates all the Part classes specified in the CMSD Map's list of Part Type objects, then looks at each Process Plan object in the CMSD Map object to determine the Part Type associated with the Process Plan. For each Process object in the Process Plan, the translator creates a QUEST Process object. This object is populated with the Part classes required and produced, and any QUEST non-machine resources required. With the Process objects created, the translator then assigns these Process objects to the different QUEST machines capable of performing those processes.

The resulting QUEST model holds all the information it needs to be able to move Part entities through their manufacturing process sequences. The rest of the model building process, including part generation, statistics collection, and machine scheduling procedures, must be performed manually. Such tasks could be further automated by adopting the complete set of CMSD classes available.

5.2 CMSD to Arena Translator

The Arena simulation tool (Rockwell 2011) provides an interactive 2D flowchart-like environment for defining a process-based simulation model. Arena provides a COM interface for automating the construction of this flowchart-like model, which has enabled the creation of a CMSD to Arena translator.

Because of Arena's process-based world view, the CMSD to Arena translator was designed to construct process flows for the different Part Type objects in a CMSD Map. Each Arena process module is created based off a CMSD Process. All the Resource objects in a CMSD map get a corresponding Resource definition within the Arena model, which can be seized and released by Arena processes.

To build the model, the CMSD to Arena translator simply loops through the list of Part Type objects and creates Arena entities. It then loops through the list of Resource objects and creates corresponding Arena resources. The translator then loops through the list of Process objects in the Process Plan object for each Part Type. Each Process object gets created as a seize-delay-release process module in the Arena flowchart, with any required resources assigned to the process.

The resultant Arena model is very basic and simply contains the sequence of processing for the different Part Types. Again, any additional modeling for part generation, statistics collection, and resource sharing/scheduling must be done manually.

5.3 CMSD to ProModel Translator

The ProModel simulation tool (ProModel 2011) provides a simple 2D interactive interface for building object-based simulation models. In the background of the ProModel simulation model is a series of data tables describing all the different objects in the model. For instance, there is a Locations table which holds the 2D locations of all the fixed resources in a model. Another table holds the definitions of different part types. Yet another table defines the processing sequences for the part types, with each row representing a process.

ProModel provides a COM interface for automatically populating these tables, which in turn populates the visual interface of the software. A programmer can hook into ProModel and populate these tables. An easier solution is to use an Excel template provided by ProModel called the ProActiveX template. This template holds templates of the data tables used by ProModel. The template also holds executable VBA code that makes all the ProModel API calls to copy the tables into ProModel from Excel. This ProActiveX tool allowed the very rapid creation of a CMSD to ProModel translator.

The construction of the CMSD to ProModel translator simply consisted of creating an exporter from CMSD into Excel. Since the CMSD Map object basically consists of lists of different object types, the translator had to simply convert each list into a spreadsheet table with appropriately named fields. For instance, each Part Type object gets mapped to an Excel row in the Entities table, with several known properties mapped to the appropriate columns. The Resource objects get broken into two separate tables based on resource type. Again, each resource is exported to its own row, and any supported properties mapped to table columns.

The processing worksheet takes a list of process sequences for different entities. This requires the translator to list out the process sequence for each Part Type object. Process sequences are defined in Process Plan objects, so the translator simply lists each Process in each Process Plan by sequence. The translator then calls on the ProActiveX code to import the spreadsheet to a functional ProModel model.

The resulting model is somewhat complete and runs, as the translator automatically assigns arrival rates to the different part types. However, the model may not be a complete, valid model, and may need further manual processing.

5.4 CMSD to FlexSim Translator

The FlexSim simulation tool (FlexSim 2011) provides a 3D environment for laying out physical, fixed resources, typically represented by machines. A user may also lay out mobile resources (task executers) like laborers or AGV's, as well as other modeling elements (queues, sources, sinks). Part routing is ac-

complished through event triggers on different model elements, by selecting a destination from a set of possibilities which are connected to the element. When a part arrives at a fixed resource, the resource samples a process time, which can be specified by a simple probability distribution, or through some logic to select an appropriate distribution to sample.

The FlexSim translator was created using an unsupported, experimental COM API provided by FlexSim, and so the functionality of the translator could vary by version of the FlexSim software. The API works by executing single script commands to modify the node tree internal structure in FlexSim. This allows the translator to create and locate modeling elements. It also populates routing connections on the elements, but does not create logic for selecting destinations for different part types. The translator also populates just a single cycle time on a machine, rather than have it dependent on part type and process step. This functionality would have to be further enhanced, either by writing some shell code that calls back to the CMSD file for sampling, or perhaps by populating routing and cycle time tables for different part types.

The models created by the FlexSim translator require much manual work in order to get running. It provides an example of a simple translator that does a simple task: build and locate machines on the floor, and copy over some simple data.

5.5 Case Study

A simple CMSD data file was created for comparing the results of the different translators. The file describes a system with three total part types: two subassemblies and one main assembly. Each part type has pre-processing steps defined, as well as assembly steps. The system also has 16 machines and 8 labor types. The data was populated in a value stream mapping software package, with an accompanying 2D layout describing the locations of the machines. A custom translator was used to create the CMSD file from the value stream mapping software.

Figure 1 below shows the resulting models in QUEST, FlexSim, and Arena. The QUEST and FlexSim models show machines laid out on a floor, in an object-based view with processes tied to the location objects. The Arena model shows more of a flow chart of processes, with resources tied to processes.

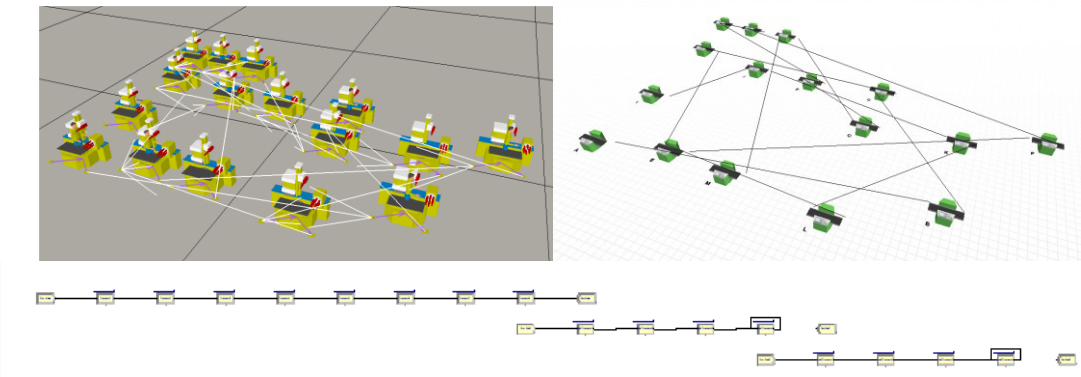

Figure 1: (clockwise from top left) QUEST, FlexSim, Arena models

Figure 2 below shows a section of the CMSD XML file describing one of the processes in the model. Note that a process is defined in sections describing parts required, parts produced, resources required, and various properties, both standard and custom.

```
- <Process>
    <Identifier>A010</Identifier>
    <Description>Process 1</Description>
  - <PartsProduced>
    - <PartType>
        <PartTypeIdentifier>Main</PartTypeIdentifier>
      </PartType>
      <PartQuantity>1</PartQuantity>
    </PartsProduced>
    <Name>Process 1</Name>
  - <operationtime>
      <Value>8</Value>
      <Unit>Mins</Unit>
    </operationtime>
  - <Property>
      <Name>Product</Name>
      <Value>Main</Value>
      <Unit>Unit</Unit>
    </Property>
  - <Property>
      <Name>Workstation</Name>
      <Value>A</Value>
      <Unit>Unit</Unit>
    </Property>
  - <ResourceRequired>
      <Description />
    - <Resource>
        <ResourceIdentifier>A</ResourceIdentifier>
      </Resource>
    </ResourceRequired>
  </Process>
```

Figure 2: Process definition in CMSD

Figure 3 below shows the beginning section of the CMSD file for this case study. The data section begins with a list of part types and resources, followed by process and process plan definitions.

Figure 4 below shows the XML description of a Placement object, which corresponds with a resource defined in the DataSection of the XML document. For the purpose of defining resource position, a Placement object basically just needs an X and Y location, as well as the LayoutElementIdentifier to tie back to the appropriate resource.

6 CONCLUSION

This paper details some of the work done to create a special purpose CMSD programming toolkit and translator software that uses the toolkit. This toolkit has been used to make the consumption of CMSD data trivial to the programmer, as all the complexity of reading a CMSD file into a useful object model is abstracted away into a single program call. Further development of this toolkit will be driven by bringing the CMSD API to fully support the CMSD standard, as well as any enhancements that make creating and consuming CMSD data easier for a programmer.

```
- <CMSDDocument>
  - <DataSection>
    - <PartType>
        <Identifier>Main</Identifier>
      </PartType>
    - <PartType>
        <Identifier>Sub1</Identifier>
      </PartType>
    - <PartType>
        <Identifier>Sub2</Identifier>
      </PartType>
    - <Resource>
        <Identifier>A</Identifier>
        <Name>A</Name>
        <ResourceType>station</ResourceType>
      </Resource>
    - <Resource>
        <Identifier>B</Identifier>
        <Name>B</Name>
        <ResourceType>station</ResourceType>
      </Resource>
    - <Resource>
        <Identifier>C</Identifier>
        <Name>C</Name>
        <ResourceType>station</ResourceType>
      </Resource>
```

Figure 3: XML descriptions of part types and resources

```
- <Placement>
  - <Location>
      <X>369.230769230769</X>
      <Y>258.33091436865</Y>
    </Location>
    <LayoutElementIdentifier>N</LayoutElementIdentifier>
  </Placement>
```

Figure 4: XML description of 2D placement of a machine resource

As these standardized translators become more powerful, their application areas will also expand. The toolkit described above can also be used for extracting useful data from simulation models into the CMSD format, where the data could theoretically be reused in a different simulation package.

Further refinement of these translators will bring further functionality to simulation models with relatively little added work required on the part of the simulation user. As the standard evolves it may be useful to define domain-specific "sub-standards" which make explicit assumptions and require very specific data. These vertically integrated standards could provide "one-click" model building functionality, requiring minimal manual user input.

The creation of a standard format for representing simulation data opens the door to making software tools that many users at many different companies can use, rather than continue to rely on customized special-purpose tools. The CMSD standard also lowers the barrier to entry for people developing their own translators. A tool developer can now focus solely on translating from one software environment in-

to the CMSD standard, and rely on using an existing translator to complete the translation into a simulation model.

DISCLAIMER

A number of software products are identified in context in this paper. This does not imply a recommendation or endorsement of the software products by the author or CCAT, nor does it imply that such software products are necessarily the best available for the purpose.

REFERENCES

DELMIA Corporation. Accessed June, 2011. http://www.3ds.com.

FlexSim Simulation Software. Accessed June, 2011. http://www.flexsim.com.

Johansson, M., B. Johansson, S. Leong, F. Riddick, and Y. T. Lee. 2008. "A Real World Pilot Implementation of the Core Manufacturing Simulation Information Model." In *Proceedings of the Simulation Interoperability Standards Organization Spring 2008 SIW Workshop*, Orlando FL: Simulation Interoperability Standards Organization.

Leong, S., Y. T. Lee, F. Riddick. 2006. "A Core Manufacturing Simulation Data Information Model for Manufacturing Applications." In *Proceedings of the Simulation Interoperability Standards Organization Fall 2008 SIW Workshop*, Orlando FL: Simulation Interoperability Standards Organization.

Microsoft Corporation. Accessed June, 2011. http://msdn.microsoft.com/en-us/library/ms753804(VS.85).aspx.

Paprotny, I., G. Mackulak, W. Zhao. 1999. "Reducing Model Creation Cycle Time by Automated Con version of a CAD AHMS Layout Design." In *Proceedings of the 1999 Winter Simulation Conference*, edited by P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, 779-783. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pegden, C. 2010. "Advanced Tutorial: Overview of Simulation World Views." In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yucesan, 210-215. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

ProModel Corporation. Accessed June, 2011. http://www.promodel.com.

Riddick, F., Y. T. Lee. 2008. "Representing Layout Information in the CMSD Specification." In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason, R. R. Hill, L. Monch, O. Rose, T. Jefferson, J. W. Fowler, 1777-1784. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Rockwell Automation Arena Simulation Software. Accessed June, 2011. http://www.arenasimulation.com.

Simulation Interoperability Standards Organization (SISO). Accessed June, 2011. http://www.sisostds.org/ProductsPublications/Standards/SISOStandards.aspx

Wu, T. C. 2005. "Definition, Analysis, and an Approach for Discrete-Event Simulation Model Interoperability." PhD thesis, Department of Industrial Engineering, Mississippi State University, Starkville, Mississippi. Accessed December, 2006. http://gradworks.umi.com/32/07/3207001.html.

AUTHOR BIOGRAPHY

JONATHAN FOURNIER is an Applications Research Engineer in the Modeling and Simulation Laboratory at the Connecticut Center for Advanced Technology. He received a Bachelor's Degree in Industrial Engineering from Western New England College in 2006. His research interests lie in modeling and simulation for manufacturing, as well as supply chain modeling, and lean manufacturing software development. His email address is fournier.jonathan@gmail.com.