On the Use of the Core Manufacturing Simulation Data (CMSD) Standard: Experiences and Recommendations

Soeren Bergmann, Steffen Strassburger
Department for Industrial Information Systems
Ilmenau University of Technology
Helmholtzplatz 3
98693 Ilmenau, GERMANY
{soeren.bergmann, steffen.strassburger} @tu-ilmenau.de

Keywords:
Core Manufacturing Simulation Data, Automatic Simulation Model Generation

ABSTRACT: *The Core Manufacturing Simulation Data (CMSD) information model is defined by SISO standards SISO-STD-008-01-2012 and SISO-STD-008-2010. The main objective of CMSD is to facilitate interoperability between simulation systems and other information systems in the manufacturing domain. While CMSD is mainly intended as standardized data exchange format, its capabilities go beyond simple data exchange. Frequently CMSD based system descriptions are used for purposes of automatic simulation model generation. In this paper, we report on practical experiences using the CMSD standard for such purposes as well as for purposes of simulation model initialization and simulation output data collection. Based on our experiences we suggest potential enhancements for a future revision of the standard.*

1. Introduction

In production and logistics, the application of commercial-off-the-shelf simulation packages (CSPs) based on discrete event simulation paradigms is commonplace. Simulation is used for planning new systems (e.g., for the prediction of system behavior) as well as for operational decision support in existing systems (e.g., for the evaluation of control alternatives).

Both application areas may require a close integration of existing information systems from the production and logistics context and CSPs. Information systems of interest include enterprise resource planning (ERP), manufacturing execution systems (MES), and production planning applications.

Scenarios requiring a close integration include the automatic simulation model generation as well as simulation model initialization.

In this article, we summarize our experiences (previously reported in [1–5]) with the usage the Core Manufacturing Simulation Data (CMSD) standard. We report about lessons learnt and suggest potential enhancements for a future revision of the standard.

The remainder of this article is structured as follows. Section 2 briefly introduces essential ideas of CMSD. Section 3 outlines our use of CMSD-based simulation model generation. Section 4 discusses CMSD-based simulation model initialization. Section 5 reports on the usage of CMSD for capturing simulation result data. Section 6 discusses lessons learnt and makes recommendations for future revisions of the standard.

2. Core Manufacturing Simulation Data

The CMSD information model is an open standard developed within the simulation interoperability standards organization (SISO). The primary objective of the CMSD information model is to facilitate interoperability between simulation systems and other information systems used in manufacturing. Towards this objective CMSD provides a data specification for the efficient exchange of manufacturing data in a simulation environment.

The CMSD standard consists of two parts. The first part uses the Unified Modeling Language (UML) representation [14]. The UML representation has been organized using packages shown in Figure 1.

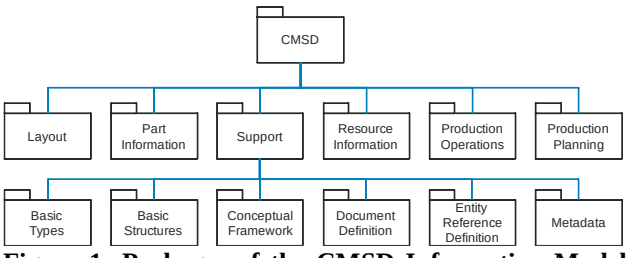


Figure 1: Packages of the CMSD Information Model [14]

The second part implements the data format in an XML schema description and is based on RelaxNG and Schematron as schema languages [15].

The CMSD standard provides data structures and an information model for the exchange of modeling information and includes classes describing jobs, parts, resources including machines and workers, process plans, shifts, etc. as well as basic layout information.

CMSDs capabilities were tested and documented in several research projects and publications [6, 7, 11, 13]. Our own work has demonstrated that CMSD is useful for the model generation [1], initialization [3], and facilitating web-based simulation usage scenarios [4]. We have investigated CMSD-based automatic model generation for both component-based simulation tools, such as Plant Simulation [1] as well as for simulation languages such as SLX [5].

3. CMSD-based simulation model generation

Different categories of input data are needed for creating simulation models of production systems. The VDI (The Association of German Engineers) classifies relevant input data into three clusters: technical, organizational, and system load data as shown in Figure 2.

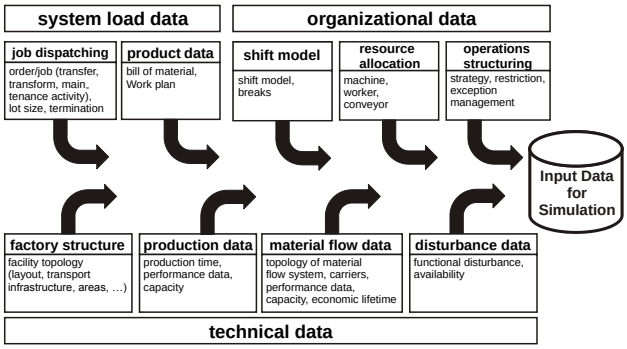


Figure 2: Input data for simulation models

The class of the technical data describes the topology and layout of the entire system as well as the properties of single system components. The organizational data specifies the operation structuring and process organization, especially working shifts models, strategies and resource allocations. Finally, the class system load data describes jobs and their properties.

While technical data and organizational data are mostly relevant for data-driven model generation approaches, the system load data focuses on the data primarily relevant for model initialization.

Although CMSD offers support for all suggested input data categories, there are sometimes multiple ways of mapping required input data to elements of the CMSD standard. This is sometimes due to missing exact matches of classes (e.g. buffers) or due to desired (but missing) properties of a certain class (e.g., capacities of resources).

With that, some degree of freedom for interpretation of the CMSD standard exists. For the use of CMSD in various model generators, a common interpretation of the CMSD standard may therefore be needed. We here report on our usage (and as such – our interpretation) of the CMSD standard.

For model generation purposes, we have most significantly relied on the *resource class* which stores information about machines and employees, the *calendar class* for storing shift and break information, and the *process plan class* which stores detailed information about the manufacturing steps that are required for different part or job types.

We further define setup conditions of machines and other resources using the *SetupDefinition class* and setup times using the *SetupChangeoverDefinition class*.

For modelling production demand, we have used the *job class* from the productions operation package. CSMD also offers other ways to model production demand, e.g., by using the *order class*, but for simplicity we have chosen to model concrete jobs.

A job contains a reference to its process plan, a release date as well as a due date and therefore carries all information required to unambiguously simulate its flow through the production.

The decision about which CMSD class to use for which purpose somewhat depends on the specific needs and capabilities of the systems to be connected. A collection of best practices might be useful for future assistance in these cases.

Our usage of CMSD, for example, is focused on job-shop production systems. We therefore rely on process plans to describe in detail all process steps, their required machines (*RessourcesRequired attribute*) with their setup state (*AllowableSetup attribute*) and also the required employee skills (*RequiredEmployeeSkill attribute*) for a process step.

On the other hand, we abstain from using the *connection class*, except for modeling the connections between input and output buffers and a machine. A flow-shop oriented production system on the other hand might be better off

using connections to describe the flow of products instead of detailed process plans for each product.

Regarding employee skill descriptions we rely on CMSDs capabilities to describe skills using the *SkillDefinition class* including skill level descriptions. We further apply a skill centric approach of describing which worker is required for a job.

While it is possible to describe the singular worker requirement of a single process step with that approach (using a reference to *RequiredEmployeeSkill)*, we found CMSD to lack built-in capabilities for modeling a more detailed distinction between skills potentially required within a process step.

Consider a scenario where different skills are required for the actual work of a process step, the skills required for setting up a resource for that process step, and the skills required for repairing a resource. For modeling such fine granular skill descriptions, we had to improvise. In our solution, skills for the actual process step are modeled in the process step, a potentially different skill for the setup is added to the setup class using a user defined property, and a potentially special repair skill was added to the resource class. While this is all possible using CMSDs extension mechanisms of "properties", it obviously requires a specific interpretation of the semantics of the newly introduces properties.

We encountered comparable issues when trying to model disturbance reaction behavior, e.g., in case of machine breakdowns. This required the introducing of the properties "MTTR" (mean time to repair) and "availability" to the *resource class*. Another addition was required for describing waste levels. We therefore added a property "reliability" to the *resource class*.

For our application scenarios, another problem was encountered by CMSD's lack of a buffer class. For queuing systems, buffers and their capacity are essential performance factors. We therefore had to model buffers using the ResourceType "other". In addition, we had to add a "capacity" property describing a buffers capacity.

Another important requirement essential for (but not limited to) job shop scenarios is the description of decision rules (e.g., sequencing rules, routing rules). Sequencing rules, for instance, are required for determining which job is to be processed next at a certain machine. Such decision rules again had to be modeled by introducing user defined properties (see section 5).

Based on our experience with the CMSD standard, we think that some of the extensions we had to introduce should be considered as core components of the CMSD standard (and therefore be included in a future revision of the CMSD standard), other extensions and interpretations could rather be clarified in the form of a collection of best practices (e.g., in the form of a SISO guidance product). Detailed recommendations are given in section 5.

Based on our interpretation of the CMSD standard, we have investigated different approaches for implementing simulation model generators. First off, a generic implementation of the chosen CMSD classes in the targeted simulators had to be created.

We created such generic implementation for Plant Simulation (a component based simulation system from Siemens PLM Software) and for SLX (a simulation language developed by Wolverine Software [10]).

The actual generation of the simulation models based on a CMSD XML representation can then be performed using different approaches. We distinguish between internal and external approaches.

Internal approaches use algorithms/scripts executed from within the simulator to read and interpret the CMSD XML description and to create the required model elements (resource, jobs, etc.). A prerequisite for this approach are appropriate interfaces for accessing XML files as well as appropriate mechanisms for script-driven creation of model elements.

In previous work we have demonstrated the feasibility of this approach for Plant Simulation focusing on typical job shop scenarios ([1] and Figure 3).

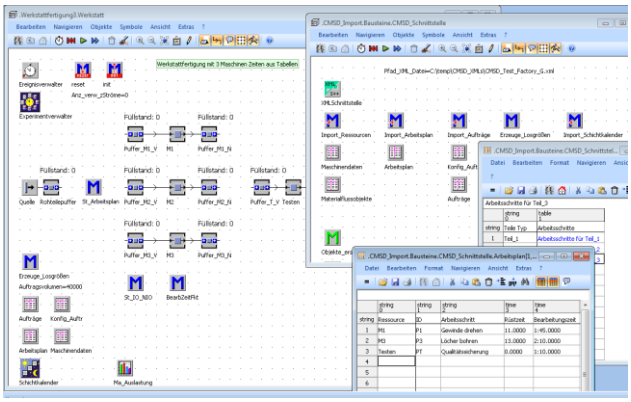


Figure 3: CMSD-based Model Generator for Plant Simulation

External approaches for simulation model generation include approaches where the actual source code of the simulation model is created externally from the simulator based on the CMSD XML description of the system. As a

proof-of-concept, we have demonstrated the use of XML Stylesheet Transformations for creating simulator source code for the SLX [5].

The work of other authors for model generators includes simulation systems such as QUEST, Arena, Pro Model, and Flexsim [7].

While all this work is very positive and emphasizes the importance of CMSD, one should also note that the use of CMSD is mostly unidirectional, i.e., from some set of data sources towards the simulation (for small exceptions – see section 4).

There is virtually no work on saving simulation models manually created in a simulation system into the CMSD format and then re-creating an equivalent simulation model automatically in a different simulator.

If at all possible, such a use of CSMD would have to be bound to

a) strict (and limiting) modelling instructions in the simulator,
b) a common interpretation of the use of CMSD elements.

In general, it can be expected that the dazzling diversity of modeling options in the chosen simulators, especially concerning dynamic model behavior, will prevent a complete and unambiguous mapping onto the CMSD standard.

Being so, CMSD still makes an important contribution for fostering interoperability between simulation systems and other IT systems in manufacturing, but it is certainly not a generic simulation model exchange format (which it also never intended to be).

3. CMSD-based model initialization

Depending on its intended use, a defined initialization of a simulation model may be a crucial requirement. Especially when used as operational decision support tool, the initialization of the simulation model must be performed in such a way that the model's internal control structures (event lists, random number generators, simulation clock, component states, etc.) reflect the current state of a real system with sufficient accuracy for forecasting purposes.

For initialization purposes, especially the system load data (see Figure 2) and the state of all resources is of interest. Table 1 summarizes the most important data categories for initialization.

Table 1: Categories of initialization data

Data about		**Example characteristics**
Resources	Machine status	Idle, working, setup, paused, failed…
	Worker	Place, working, paused, …
	Conveyor	Idle, working, paused, failed, speed, type, number
Job		Process step, state, scrap percentage, type …
Part		Place, state
System time		

Data on the states of resources shall be discussed first. Concerning machines, the active setup of the machine and its current working state are particularly important. Fundamentally, we can distinguish six main working states of a machine: idle, busy, setup, broken/failed, paused, and under maintenance. The information which specific job currently occupies the machine is only of secondary interest, as this can typically be modeled as a property of the job.

While machines are typically immobile, we have to distinguish other resources like workers and conveyors, for which the current location can also be of relevance.

Workers have partially other relevant states as other resources. Similar to the machines they have an attribute "working state", but it can have other values. While "in movement" is a valid status for a worker, "failed" is not. Furthermore, workers are usually mobile resources, so they have a current location (often at a machine). When "in movement" they should have a destination and an arrival time.

Conveyor is a class of resource which can have quite heterogeneous properties depending on the type of conveyor. Depending on the level of detail in the simulation model, in the simplest case it can be treated like a machine. Other parameters, such as current speed, acceleration, type, location, and number of carriers can be important if they are represented in the simulation model.

The central element for initialization of simulation models are the jobs in the system, as they represent the dynamic objects of the physical system. Without their accurate reproduction in the model, we cannot use it as a tool for operational decision support. The basic requirement for initializing a job appropriately is to know its current process step and its processing status. It also has to have knowledge about its process plan, e.g., its machine order.

From these two facts crucial information for the simulation can be derived: If a job is at a certain process

step (say 7) and has a certain state (say blocked) we can derive that it is located in a buffer in front of machine 7. Similarly, if its state is "started" we can derive that it is being processed at a certain machine.

Table 2: CMSD classes used for initialization and relevant attributes (excerpt from [3]).

Data	CMSD Class	Relevant Attributes
Machine state	Resource (type = machine or station)	CurrentStatus: ResourceStatus
		AssociatedResource: ResourceReference (Worker)
Worker	Resource (type = employee)	CurrentStatus: ResourceStatus
		Property - current location (LocationDefinition)
Conveyor	Resource (type = carrier, conveyor, "power and free" or trans-porter)	CurrentSetup: SetupDefinitionReference
		CurrentStatus: ResourceStatus
		AssociatedResource: ResourceReference (Worker)
		Property - current speed, acceleration, and type, location and number of carriers
Job	Job	Status: JobStatus
		Priority: String
		ActualEffort: JobEffortDescription
		PlannedEffort: JobEffortDescription
	JobEffort-Description	DueDate / ReleaseDate: TimeStamp
		StartTime / EndTime: TimeStamp
		ProcessPlan: ProcessPlanReference
		CurrentProcessPlanStep: ProcessReference
		MaintenancePlan: MaintenancePlanReference
		CurrentMaintenancePlan-Step: MaintenanceProcess-Reference
		Property - remaining pro-cessing times (double) [%]
Schedule	Schedule	StartTime / EndTime: Timestamp
		ScheduleItem: ScheduleItem
	ScheduleItem	AssociatedJob: JobReference
Part	Part	ProductionStatus: PartProductionStatus
		Location: LocationDefinition

From the states and conditions discussed above, a certain set can be used for initialization quite easily. This is especially true for all enumerated data types which merely describe a state of an element (e.g., machine state "idle").

Other data, like the current status of already started jobs (including maintenance or repair jobs) can be quite difficult to capture from the real system and to map into the simulation model state. First of all, this data will most likely not be explicitly available from the real system. Rather, if we want to know a remaining process time, we will most likely only be able to determine a job's starting time and its planned processing time. From this we may be able to estimate its remaining processing time. Still, it may be difficult to appropriately integrate this information into the simulation system.

The CMSD standard offers a variety of classes which can be used for representing the data relevant for initialization. We suggest the usage of the classes *Resource*, *Part*, *Job*, *JobEffortDescription*, *Schedule*, *ScheduleItem* and *ProcessPlan*. Table 2 exemplifies our suggested use.

The developed model generators described in the previous section are capable of performing model initialization based on the attributes indicated in Table 2.

Sometimes user-defined attributes ("properties" in the CMSD terminology) had to be used when CMSD offered no predefined attributes suitable for the required purpose. This applies, for instance, to the current location of workers or the remaining processing time of jobs.

Further enhancements are needed for representing the current state of conveyors, but are beyond the scope of discussion here. Details can be found in [3].

While the suggested extensions using properties are designed to increase the accuracy of initializing simulation models, a backward compatibility is easily maintained, as initialization routines not capable of handling a certain property will still be able to perform basic initialization (ignoring additional properties), even if initialization is then performed at a lower degree of accuracy.

4. CMSD-based output analysis

4.1 Capturing simulation result data in CMSD

CMSD-based simulation model generation (section 2) and initialization (section 3) so far have considered how data from external data sources can be transferred into data usable in the simulation.

Our work on CMSD-based output analysis goes the opposite direction. Here, we investigated, if CMSD is capable of capturing simulation result data appropriately and what can be done towards its analysis.

Simulation output data analysis is a well-studied domain and must be carried out considering certain statistical rules (replications for non-deterministic models, etc.) [12].

The type of desired output values is often highly dependent on the simulation problem at hand. Typical key performance indicators for manufacturing systems include average cycle times, setup times, adherence to delivery dates, resource utilization, etc.

To provide a great degree of flexibility for output data analysis, simulation result data must be captured in a way that all possible information needed for analyzing the simulated system are contained.

We therefore suggest an abstraction level in which all information potentially relevant for output analysis can be represented adequately. Towards this, we suggest to capture simulation output analogous to data a real production data acquisition (PDA) system would capture.

In PDA, typically data about events on jobs, resources etc. are collected. Events typically occur as a result of a status change of an object, e.g., a job starts working on a machine and is allocated a worker, or a machine fails. All these kinds of events can be described by a timestamp, an identifier, and, if necessary, references to related objects, like jobs or resources.

The CMSD data structure most appropriate for these purposes is the *event class* (Figure 4). The *event class* is part of the basic structure package which itself is part of the support package. The *event class* according to the CMSD standard is only used by the *JobEffortDescription class* located in the Production Operations Package.

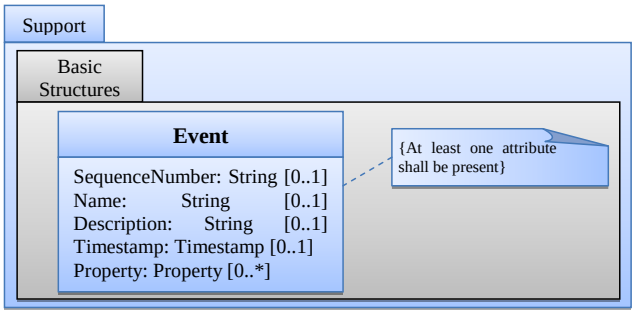


Figure 4: The CMSD Event Class, as part of the Support/BasicStructures Package

According to CMSD, the *event class* provides a means to plan for or record the occurrence of some phenomenon, condition, or state that is relevant to production activities. It can also typically be used to describe the actual effort that occurred when processing a job. We suggest enhancing this usage for more detailed simulation result documentation purposes.

The Event Class has five attributes, from which all but the attribute *description* are used in our approach.

Firstly, the attribute *SequenceNumber* is used to order events in a logical order. Every event has a unique number.

Secondly, the attribute *Name* classifies the type of the recorded event. We suggest an enumeration of possible values. These values mostly relate to state information and include values such as start setup, start work, end work, machine broken, machine repaired, etc.

Thirdly, the attribute *Timestamp* contains the date and time when the event occurred. The representation is defined according to ISO 8061.

Finally, we use at least one event type specific *property* attribute. This *property* is used to record a relation of the event to one or more objects it refers to, e.g., a worker or machine.

When events are recorded that involve jobs (that is their original purpose), the involved job is identifiable through the hierarchy of the CMSD document, because the event class is used inside a *JobEffortDescription* of a *Job*.

When event information must be recorded that does not directly relate to a job, this job-centric use of the event class may be problematic. An example for this is a machine breakdown while no job is currently being processed on it.

If this type of event is considered relevant for result evaluation, we have different options to circumvent this limitation in CMSD. A simple way for managing such events is to co-locate the event with the last known job on this machine. For this alternative we do not need any extra property, but we are “extending” the intended use of the event class inside the job class.

A second way to deal with this problem is to convert from a job oriented view of events to a machine oriented view as it would occur in real PDA. This could be done for all events or only for special events. This alternative is logically correct, but is problematic as resources like machines in the CMSD standard do not have an Event

attribute. For this approach, we would have to use a user-property to extend the CMSD standard, e.g., a reference property to the Event Class.

For our tests of web based simulation output analysis, we have used a third (and highly pragmatic) approach by introducing a dummy job as a container for all non-job-specific events.

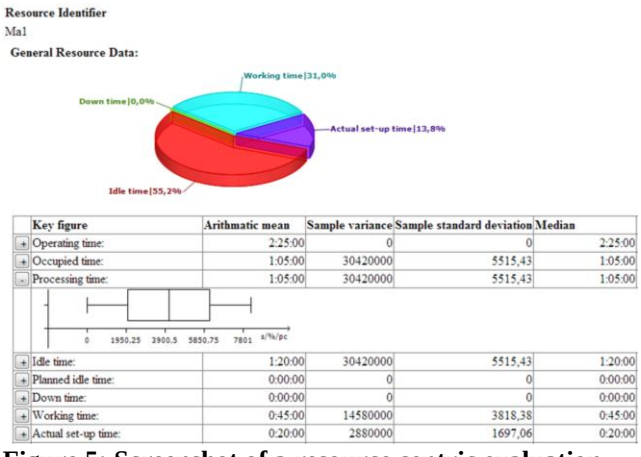
4.2 Statistics Monitor

The objective of the statistics monitor developed within our framework for web-based simulation was to compute and visualize key performance indicators for the simulated systems based on the event logs added to the CMSD files during the simulation.

The statistics monitor allows two modes of evaluation:

1) Evaluation of a single CMSD result file.
2) Evaluation of multiple CMSD result files obtained from different simulation runs that were previously defined, e.g., for implementing replications.

Depending on the mode of operation, different visualizations and performance indicators can be computed. Gantt-Charts visualizing job processing are an example of a visualization useful for the single CMSD file analysis. Multiple CMSD file analysis allows the computation of typical statistical measures like mean values, confidence intervals, standard deviations, etc. Different views (resource centric – see Figure 5, job centric) can be defined.



Key figure	Arithmatic mean	Sample variance	Sample standard deviation	Median
Operating time:	2:25:00	0	0	2:25:00
Occupied time:	1:05:00	30420000	5515,43	1:05:00
Processing time:	1:05:00	30420000	5515,43	1:05:00
Idle time:	1:20:00	30420000	5515,43	1:20:00
Planned idle time:	0:00:00	0	0	0:00:00
Down time:	0:00:00	0	0	0:00:00
Working time:	0:45:00	14580000	3818,38	0:45:00
Actual set-up time:	0:20:00	2880000	1697,06	0:20:00

Figure 5: Screenshot of a resource centric evaluation

All performance indicators are computed based on aggregated event data, i.e., performance indicators such as cycle time and delay are obtained by post-processing the event logs. In the same way, statistical values for groups of entities (e.g., the mean value of the cycle time of all jobs, the maximum setup time in front of a machine, etc.) are computed.

4.3 Animation

We further investigated the applicability of the created CMSD event logs for a post-processed animation of simulation runs [2].

The basic idea here was to use the layout information contained in the CMSD layout package to create a static scene indicating resource locations using predefined resource symbols.

The animation of the scene is then performed based on event information from the CMSD events. This includes the state change of resources (working/idle/broken) as well as the movement of jobs and workers in the system.

Currently, a proof-of-concept implementation for 2D animation based on the HTML 5 Canvas element and the JavaScript Frameworks JQuery and KineticJS has been implemented (see Figure 6 and [2]).

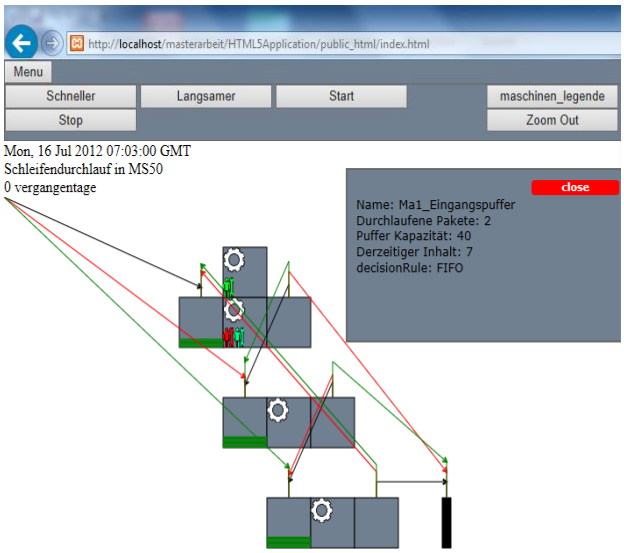


Figure 6: Sample screenshot of animation generated from a CMSD file

From the CMSD point of view, no problems or additional requirements towards animation were encountered. All dynamics that is needed for a generic animation can be expressed in CMSD events. Although they are not animation specific, it is possible to automatically visualize a basic animation of the simulation run based on that.

It should be noted, though, that CMSD is no graphics exchange format. Therefore it does not contain any

geometrical scene description, but rather basic location and shape information. Also, CMSD events in the used form are no complete animation trace description, like it is known from specialized animation systems like Proof Animation [8, 9] or from visualization systems known from the Virtual Reality domain.

5. Lessons Learnt & Recommendations

In different application scenarios we have successfully verified the suitability of CMSD for modeling complex production systems. We have successfully used CMSD for extracting data from enterprise resource planning systems such as SAP ERP and have automatically generated simulation models in different simulation systems based on the extracted information.

Our experience shows that there is sometimes room for different interpretations about the intended use of some CMSD classes. These interpretations can sometimes constrain the exchange of data between different IT systems and/or actors. Towards that, we suggest the development and release of reference implementations exemplifying the intended use of CMSD in certain scenarios (e.g., for job shop and flow shop production systems).

In our work, we rather frequently had to use the built-in extensibility mechanism of CMSD. Virtually every CMSD class can be extended using user-defined properties. While this feature obviously increases the flexibility of CMSD, each use of a property introduces a user-specific enhancement of the standard, which may create incompatibilities between different users, application scenarios, or implementations that do not know how to use this property.

Table 3 documents those user properties we created that we consider candidates for an inclusion as standard properties in a future release of the CMSD standard.

Especially the attribute "capacity" which we needed to model buffer capacities is an element that should be considered crucial for any resource, not only buffers.

Also decision and routine rules are candidates which we consider important for modelling sequencing and routing logic that go beyond simple "First-In-First-Out" style default behavior.

Availability, MTTR, and reliability are important enhancements for describing the behavior of resources in conjunction with breakdown and repair. The importance of setupSkills and repairSkills was discussed in section 2.

Table 3: Used properties that are suggested for inclusion in a future version of CMSD standard.

Property Name	Extended Class	Data Type/ Allowed Values	Description/ Intention
capacity	Resource {Resource-Type= other; buffer only}	Integer (Values <0 for infinite capacity)	Capacity of a buffer (could extend any resource)
decision-Rule	Resource {Resource-Type= other; buffer only}	Enumeration type "Decision-Rules"	Sequencing rule applied to the exit of a buffer
routing-Rule	Resource {Resource-Type= other; buffer only}	Enumeration type "Routing-Rules"	Routing rule applied to the exit of a buffer
availability	Resource {Resource-Type= station OR machine}	Decimal (>=0 ; <=100)	Availability in %
MTTR	Resource {Resource-Type= station OR machine}	Decimal (>=0)	Mean Time to Repair after a failure of a resource
reliability	Resource {Resource-Type= station OR machine}	Decimal (>=0 ; <=100)	Waste rate of a resource in %
setupSkill	Setup-Definition	Skill-Reference	Reference to specific skill required for setup
repairSkill	Ressource {Resource-Type= station OR machine	Skill-Reference	Reference to a specific skill required for repair

Beyond these properties, some existing enumeration types had to be extended for our purposes. Table 4 lists those extensions that we also consider candidates for an inclusion in a future release of the CMSD standard. We especially think that the addition of the value "buffer" to the enumerated data type *ResourceType* is a crucial addition missing in the original CMSD standard.

A further useful extension concerning the usability of events could be made by allowing references to Events. This would, among others, require the addition of "EventReference" to the enumerated data type *ReferenceTypeName*. In addition, the *event class* should

be supplemented with an attribute "EventType" based on the suggested enumerated data type "EventType" (see table 4).

Table 4: Extended/created enumerated data types that are suggested for inclusion in a future version of CMSD standard.

Enumerated Data Type (*indicates suggested new type)	Values (* indicates suggested extension)
ResourceType	carrier
	conveyor
	crane
	employee
	fixture
	machine
	path
	powerAndFree
	station
	tool
	tranporter
	buffer*
	other
ResourceStatus	busy
	idle
	broken
	underMaintenance
	unknown
	setup*
	paused*
DecisionRules*	FIFO*
	LIFO*
	KOZ*
	LOZ*
	SST*
	HCM*
	Slack*
	Random*
	…
RoutingRule*	SST*
	roundRobin*
	Random*
	…
EventType*	released*
	complete*
	start work*
	finish work*
	start setup*
	broken*
	repaired*
	start transportation*
	finish transportation*
	…

A final issue concerns the usability of stochastic distributions. Although definition and use of distributions and their parameters is possible using the *Distribution class* and the *DistributionParameter class*, there are no predefined distributions in CMSD. While a distribution can be easily modeled by each modeler, there is no prescription on the naming conventions for distributions and their parameters. This issue does not necessarily require an enhancement of the CMSD standard, but could be solved by providing reference classes for the most common distribution functions.

6. Summary

This paper presented our experience with the practical application of the CMSD standard. While generally very successful, we also found that CMSD on certain occasions leaves room for different interpretations and different styles of usage.

Most of these issues could be solved by providing reference implementations and best practice documentations. This could take the form of SISO guidance products, e.g., for documenting the use of CMSD for certain production types, or for documenting the unambiguous use of certain classes (e.g., relating to distribution functions).

On some occasions, we found items to be missing in the CMSD standard. This mostly related to attributes of classes or enumeration types. While CMSD's extensibility mechanism in most of these cases allowed a standard-compliant extension, e.g., by adding user defined properties to a class, some of these extensions could be a worthwhile addition to a future revision of the CMSD standard.

One of the core items that we would like to put forward for such a revision is the inclusion of the type "buffer" to the *ResourceType* enumeration, and the inclusion of the attribute "capacity" to the *resource class*.

We also consider the suggested extensions for describing the behavior of resources in conjunction with breakdown and repair a crucial element needed in almost any manufacturing simulation.

When thinking about extending CMSD to become a (at least partial) model exchange format for manufacturing simulations, more thought must also be given on modeling dynamic behavior in CMSD. The suggested inclusion of decision and routing rules is one step into that direction.

Finally, the suggested extended use of the event class could open new options for using CMSD as simulation trace format.

7. References

[1] Bergmann, S., Fiedler, A., and Strassburger, S. 2010. Generierung und Integration von Simulationsmodellen unter Verwendung des Core Manufacturing Simulation Data (CMSD) Information Model. Generation and Integration of Simulation Models Using the Core Manufacturing Simulation Data (CMSD) Information Model. In *Tagungsband der 14. ASIM-Fachtagung Simulation in Produktion und Logistik - Integrationsaspekte der Simulation: Technik, Organisation und Personal*. ASIM-Mitteilung 131. Technische Informationsbibliothek u. Universitätsbibliothek; KIT Scientific Publ, Hannover, Karlsruhe, 461–468.

[2] Bergmann, S., Parzefall, F., and Straßburger, S. 2014. Webbasierte Animation von Simulationsläufen auf Basis des Core Manufacturing Simulation Data (CMSD) Standards. Web-based Animation of Simulation Runs using the Core Manufacturing Simulation Data (CMSD) Standard. In *22. Symposium Simulationstechnik (ASIM 2014)*. ASIM Mitteilung 151. ARGESIM / ASIM, Wien, 63–70.

[3] Bergmann, S., Stelzer, S., and Strassburger, S. 2011. Initialization of Simulation Models Using CMSD. In *Proceedings of the 2011 Winter Simulation Conference (WSC 2011)*. IEEE, Piscataway, NJ, 2223–2234. DOI=10.1109/WSC.2011.6147934.

[4] Bergmann, S., Stelzer, S., and Strassburger, S. 2012. A New Web Based Method for Distribution of Simulation Experiments Based on the CMSD Standard. In *Proceedings of the 2012 Winter Simulation Conference (WSC 2012)*, 3057–3068. DOI=10.1109/WSC.2012.6464985.

[5] Bergmann, S., Stelzer, S., Wuestemann, S., and Strassburger, S. 2012. Model Generation in SLX using CMSD and XML Stylesheet Transformations. In *Proceedings of the 2012 Winter Simulation Conference (WSC 2012)*, 3046–3056. DOI=10.1109/WSC.2012.6464981.

[6] Bloomfield, R., Mazhari, E., Hawkins, J., and Son, Y.-J. 2012. Interoperability of Manufacturing Applications Using the Core Manufacturing Simulation Data (CMSD) Standard Information Model. *Computers & Industrial Engineering* 62, 4, 1065–1079.

[7] Fournier, J. 2011. Model Building with Core Manufacturing Simulation Data. In *Proceedings of the 2011 Winter Simulation Conference (WSC 2011)*, 2219–2227.

[8] Henriksen, J. O. Adding Animation to a Simulation Using Proof. In *Proceedings of the 2000 Winter Simulation Conference*, 191–196.

[9] Henriksen, J. O. 1999. General-Purpose Concurrent and Post-Processed Animation with Proof. In *Proceedings of the 1999 Winter Simulation Conference*, 176–181.

[10] Henriksen, J. O. 1999. SLX - The X is for eXtensibility. In *Proceedings of the 1999 Winter Simulation Conference*, 167–175.

[11] Johansson, M., Leong, S., Lee, Y. T., Riddick, F., Shao, G., Johansson, B., Skoogh, A., and Klingstam, P. 2007. A Test Implementation of the Core Manufacturing Simulation Data Specification. In *Proceedings of the 2007 Winter Simulation Conference*. *December 9 - 12, 2007, Washington, DC, U.S.A.* ACM, IEEE, New York, NY, 1673–1681.

[12] Law, A. M. 2014. *Simulation Modeling and Analysis*. McGraw-Hill series in industrial engineering and management science. Mcgraw Hill Book Co.

[13] Lee, Y.-T. T., Riddick, F. H., and Johansson, B. 2011. Core Manufacturing Simulation Data – a Manufacturing Simulation Integration Standard: Overview and Case Studies. *International Journal of Computer Integrated Manufacturing* 24, 8, 689–709.

[14] Simulation Interoperability Standards Organization. 2010. *Standard for: Core Manufacturing Simulation Data - UML Model*. Core Manufacturing Simulation Data Product Development Group, SISO-STD-008-2010. http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=31457.

[15] Simulation Interoperability Standards Organization. 2012. *Standard for Core Manufacturing Simulation Data – XML Representation*. Core Manufacturing Simulation Data Product Development Group, SISO-STD-008-01-2012. http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=36239.

Author Biographies

SÖREN BERGMANN holds a Doctoral and Diploma degree in business information systems from the Ilmenau University of Technology. He is a member of the scientific staff at the Department for Industrial Information Systems. Previously he worked as corporate consultant in various projects. His research interests include generation of simulation models and automated validation of simulation models within the digital factory context. His email is soeren.bergmann@tu-ilmenau.de.

STEFFEN STRASSBUGER is a professor at the Ilmenau University of Technology and head of the Department for Industrial Information Systems. Previously he was head of the "Virtual Development" department at the Fraunhofer Institute in Magdeburg, Germany and a researcher at the DaimlerChrysler Research Center in Ulm, Germany. He holds a Doctoral and a Diploma degree in Computer Science from the University of Magdeburg, Germany. He is further an associate editor of the Journal of Simulation. His research interests include distributed simulation, automatic simulation model generation, and general interoperability topics within the digital factory context. He is also the Vice Chair of SISO's COTS Simulation Package Interoperability Product Support Group. His web page can be found via www.tu-ilmenau.de/wi1. His email is steffen.strassburger@tu-ilmenau.de.