Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds

MODEL GENERATION IN SLX USING CMSD AND XML STYLESHEET TRANSFORMATIONS

Sören Bergmann
Sören Stelzer
Sascha Wüstemann
Steffen Strassburger

Ilmenau University of Technology
Helmholtzplatz 3
98684 Ilmenau, GERMANY

ABSTRACT

This article introduces a novel methodology for automatic simulation model generation. The methodology is based on the usage of XML stylesheet transformations for generating the actual source code of the target simulation system. It is therefore especially well-suited for all language-based simulation systems. The prerequisite for using the methodology is an appropriate representation of the system under investigation in the Core Manufacturing Simulation Data (CSMD) format. The applicability of our methodology is demonstrated for the simulation language SLX as well as for the visualization system Proof Animation.

1 INTRODUCTION

Discrete-event simulation is used within many different disciplines and application areas. In the area of production and logistics, it is a well-accepted tool for the planning, evaluation, and monitoring of relevant processes (VDI 3633-1). Fowler and Rose (2004) have discussed future challenges for modeling and simulation of complex production systems. Among others, they identified the reduction of the time and effort for simulation studies as well as the integration of simulation with the real production system as future research areas.

As discussed in previous work, an approach for a reduction of the time and effort in simulation studies is the idea of automatic model generation and initialization (Bergmann and Strassburger 2010; Bergmann, Fiedler, and Straßburger 2010; Bergmann, Stelzer, and Strassburger 2011). So far, these approaches have focused on component-based simulators like Plant Simulation (Siemens 2012). The common idea for generating models in such tools is to dynamically create the required model elements based on an external data format describing the model structure. A suitable data format for manufacturing systems is the Core Manufacturing Simulation Data (CMSD) standard (see Section 2). It is important to note that the model generation here is performed from within the simulator itself, i.e., a suitable model generation script is executed in the simulator, reads the CMSD input data file, and accordingly generates the appropriate model elements. The basic types of model elements must be predefined in the component libraries of the simulation system.

In this paper we present a completely novel methodology for automatic model generation which is applicable for language-oriented simulation tools, such as SLX (Henriksen 1999) or Desmo-J (Desmo-J 2012).

For these tools we suggest an approach which generates the actual code of the simulation model based on eXtensible Markup Language (XML) stylesheet transformations. The advantage of this approach is increased flexibility and easy adaptability to other simulation languages.

The remainder of this paper is structured as follows: In Section 2 we introduce the CMSD standard and discuss its current application. In Section 3, we introduce the XML stylesheet transformation language named eXtensible Stylesheet Language (XSL) Transformation (XSLT) which is used for the model generation. In Section 4, we first introduce the simulation and visualization tools used for demonstrating our methodology, namely, SLX and Proof Animation. We then present a prototypical implementation of a CMSD-based XSLT model generator for both tools. Section 5 critically reviews the achievements and documents possible areas of future work.

2 CORE MANUFACTURING SIMULATION DATA INFORMATION MODEL

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 manufacturing applications. Towards this objective it provides a data specification for the efficient exchange of manufacturing data in a simulation environment (SISO 2010).

The CMSD standard consist of two parts. The first part uses the Unified Modeling Language (UML) representation. The UML representation has been organized using packages shown in Figure 1. The second part implements the data format in an XML schema description and is based on RelaxNG and Schematron as schema languages. For more detailed information about CMSD, see SISO (2010), SISO (2011) and Johansson et al. (2007).

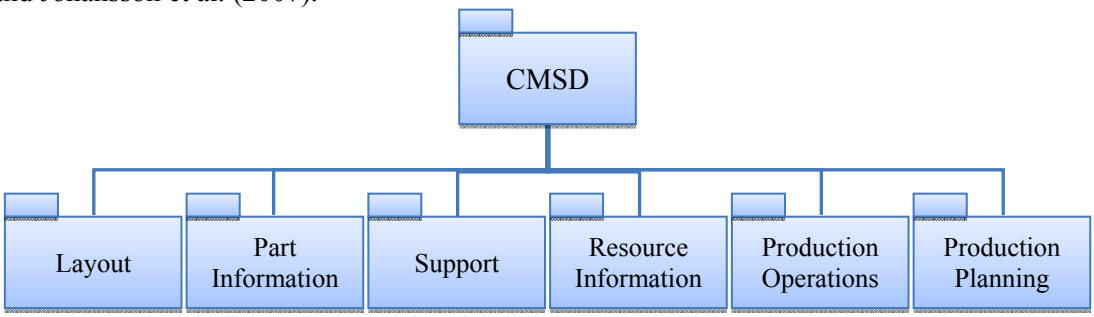

Figure 1: The packages of the CMSD Information Model (SISO 2010)

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.

The capabilities of CMSD were first demonstrated in a research project (FACT), which focused on developing new and modified production systems (Johansson et al. 2007). Furthermore it was demonstrated that CMSD is useful for the model generation (Bergmann, Fiedler, and Straßburger 2010) and initialization (Bergmann, Stelzer, and Strassburger 2011) when component-based simulation tools are used, e.g., Plant Simulation.

In this paper we focus on how to use CSMD in the context of language-based simulation tools like SLX. As the suggested method is based on using XML stylesheet transformation to transform CMSD data into SLX source code, the next section gives a brief overview about the basics of XML and XML stylesheet transformation.

3 XML STYLESHEET TRANSFORMATION

XML is the state-of-the-art standard for representing data and information in many different application areas. XML is a proven technology where data from different applications must be stored, retrieved, shared or processed. It is "playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere" (W3C 2012).

In the last few years, hundreds of XML-based languages and data formats have been developed including CMSD, RSS, SOAP, and XHTML. In addition many schema languages, related specifications and programming interfaces have been implemented, e.g., XML Schema, RelaxNG, XSLT, XSL-FO, SAX.

A common requirement in many practical applications is the transmutation of one or more XML documents into one or more target documents. The target format of these documents may be a different XML encoding or a non-XML format. This conversion process is called XML transformation and is typically described using the XSL. XSL is actually an entire family of languages for describing the transformation and rendering of XML documents. The XML transformation process has two aspects. It consists of "first, constructing a result tree from the XML source tree and second, interpreting the result tree to produce formatted results suitable for presentation on a display, on paper, in speech, or onto other media. The first aspect is called tree transformation and the second is called formatting." (W3C 2006). An example language for specifying the visual formatting of an XML document is XSL Formatting Objects (XSL-FO). For the scope of this paper, we are only concerned with a language for the tree transformation, in our case we chose XSLT. XSLT 2.0 represents a significant increase in the capability of the language compared to XSLT 1.0; for details see W3C (2007).

XSLT is a declarative, XML-based language for the transformation of XML documents. In this transformation, new documents are created based on the content of existing XML documents and a set of XSLT rules. The basic idea of this process is shown in Figure 2. It is interesting to note that the original document is not changed and that the result documents can be serialized in XML syntax (with their own schema) or in arbitrary other formats, e.g., HTML or plain text. Further principles are that:

- XSLT is based on the logical tree structure of an XML document,
- the transformation rules are defined as XSLT stylesheet (also defined as XML),
- the addressing of individual subtrees for the transformation uses XPath (W3C 2010), and
- an XSLT processor (e.g., SAXON (SAXON 2012)) is used as the XSLT template processing engine.

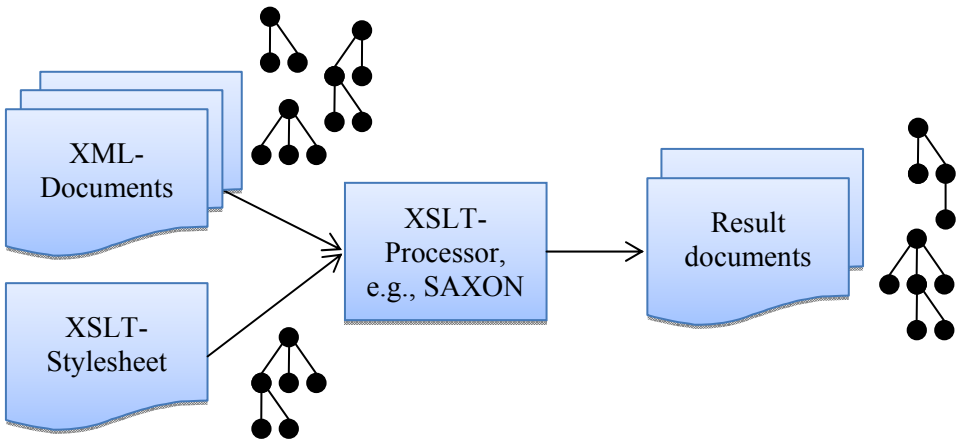

Figure 2: Process flow of eXtensible Stylesheet Language Transformations

The XSLT transformation rules, called the template, have an XPath-based pattern describing the required nodes and a content part that determines the result tree (Figure 3).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">

   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

   <xsl:template match="/rootNode">
      <xsl:for-each select="//subNode">
          <xsl:value-of select="subSubNode"/>
      </xsl:for-each>
   </xsl:template>

</xsl:stylesheet>
```

Required nodes (XPath Pattern)

Content part

Figure 3: Snapshot of a simple XSLT file

In the content part, the multitude of allowed elements ranges from simple text outputs to powerful features for analyzing and transforming the elements of the source document. It also includes control structures (e.g., repetitions, conditions) and allows the execution of external function calls (e.g., JAVA methods). In the following section, we describe a prototype which uses XSLT for both the generation of SLX and Proof Animation source code and its execution in the respective tools.

4 PROTOTYPE OF A CMSD-BASED MODEL GENERATOR FOR SLX AND PROOF ANIMATION USING XML STYLESHEET TRANSFORMATION

Simulation Language with eXtensibilities (SLX) is a discrete-event simulation tool for Windows (Henriksen 1999). SLX is a language based on a layered, inverted pyramidal software architecture. The SLX language has a C-like syntax and is an object-based language, i.e., it uses classes and objects. SLX allows simulation modeling with a process-oriented world view. Objects can be active and passive, offering the user the full flexibility to model resources, parts, jobs, etc.

The process-oriented world view is supported by simulation-specific statements, like instructions for modeling concurrency, coordination, and synchronization of processes (Schulze and Henriksen 2002). Most notably, SLX offers a generalized “wait until” statement with which processes can be logically delayed until an arbitrary boolean expression becomes true (Henriksen 2009). With SLX 2.0, the object orientation in SLX has been greatly enhanced and now includes features like inheritance, information hiding, and polymorphism.

Proof Animation on the other hand is a family of software for post-processed or concurrent visualization of simulation models (Henriksen 1997). Proof Animation uses two kinds of input files: a layout file and an animation trace file or animation trace stream. Typically, SLX and Proof Animation are used in conjunction, as SLX offers comprehensive functionalities for creating trace files or streams during a simulation run, but offers no visualization itself. The Proof Animation layout file contains information about

- the geometry/appearance of the modeled system,
- classes (mainly used for dynamic elements moving through the animation),
- paths (on which objects created in the animation will move),
- bar charts, plots, and messages (further dynamic elements of a layout which can be changed during a visualization), and
- views (which define sections of the layout to be displayed).

The automatic model generation for SLX and Proof Animation is based on the assumption that all information which is required to describe the system under investigation (and optionally its initial state) is captured in a CMSD XML file. In a practical setting, this data would be extracted from planning systems

and/or manufacturing IT systems like ERP or MES. In our test scenario, the CMSD files are generated by a web-based front end. CMSD and the web front end were also used in previous work, e.g., for the initialization of automatically generated simulation models (Bergmann, Stelzer, and Strassburger 2011). An excerpt from one of the CMSD XML files used can be seen in Figure 4. It details basic information about a resource, its location in the layout, and its current setup state.

```
<Placement>
    <LayoutElementIdentifier>Layout_Ma3</LayoutElementIdentifier>
    <Location> <X>338</X>   <Y>350</Y> </Location>
</Placement>
…
<LayoutObject>
    <Identifier>Layout_Ma3</Identifier>
    …
    <AssociatedResource>    <ResourceIdentifier>Ma3</ResourceIdentifier>
    </AssociatedResource>
    ...
</LayoutObject>
…
<Resource>
    <Identifier>Ma3</Identifier>
    <Description>drilling</Description>
    <ResourceType>machine</ResourceType>
    …
    <Name>drill</Name>
    <CurrentStatus>idle</CurrentStatus>
    <CurrentSetup>
        <SetupDefinitionIdentifier>SD_3_A</SetupDefinitionIdentifier>
    </CurrentSetup>
     …
</Resource>
```

Figure 4: Snapshot of an example CMSD.xml file

As a prerequisite for the model generation, a generic SLX implementation for the CSMD classes had to be created. This implementation is contained in the CMSD.slx source file. It implements a mapping of the CMSD classes into SLX 2.0 classes, including the class hierarchy and the attributes specified in the CMSD standard. The generation of these classes was also partially automated, as it is derived from the CMSD schema document. This is only done once, however, and does not need to be changed as long as the CMSD standard remains unmodified. For each class a generic dynamic behavior was modeled manually according to certain conventions regarding a CMSD-compatible modeling philosophy. It is important to note that this CMSD.slx file can be used repeatedly and only has to be modified when essential changes to the CMSD standard occur. The code generated by the XSLT-based model generation process will make uses of the classes defined in CMSD.slx (e.g., by instantiating classes according to the data from the supplied CMSD file).

The workflow for the CMSD-based model generation using XSLT for SLX and Proof Animation is shown in Figure 5. Only the blue parts of the figure are discussed in this paper; the rest is included only for reasons of completeness. The process of simulation model generation can be divided into four steps, as marked in the figure.

In the first step, an XSLT 2.0 processor, e.g., Saxon, transforms the CMSD XML file into two result files, namely, the SLX source code of the model and a Proof Animation layout. This transformation process is based on two XSLT templates, describing the transformation of the CMSD input into source code of an SLX model and a layout file for Proof Animation, respectively. The XSLT templates are named CMSD_to_SLX.xsl and CMSD_to_P5.xsl.

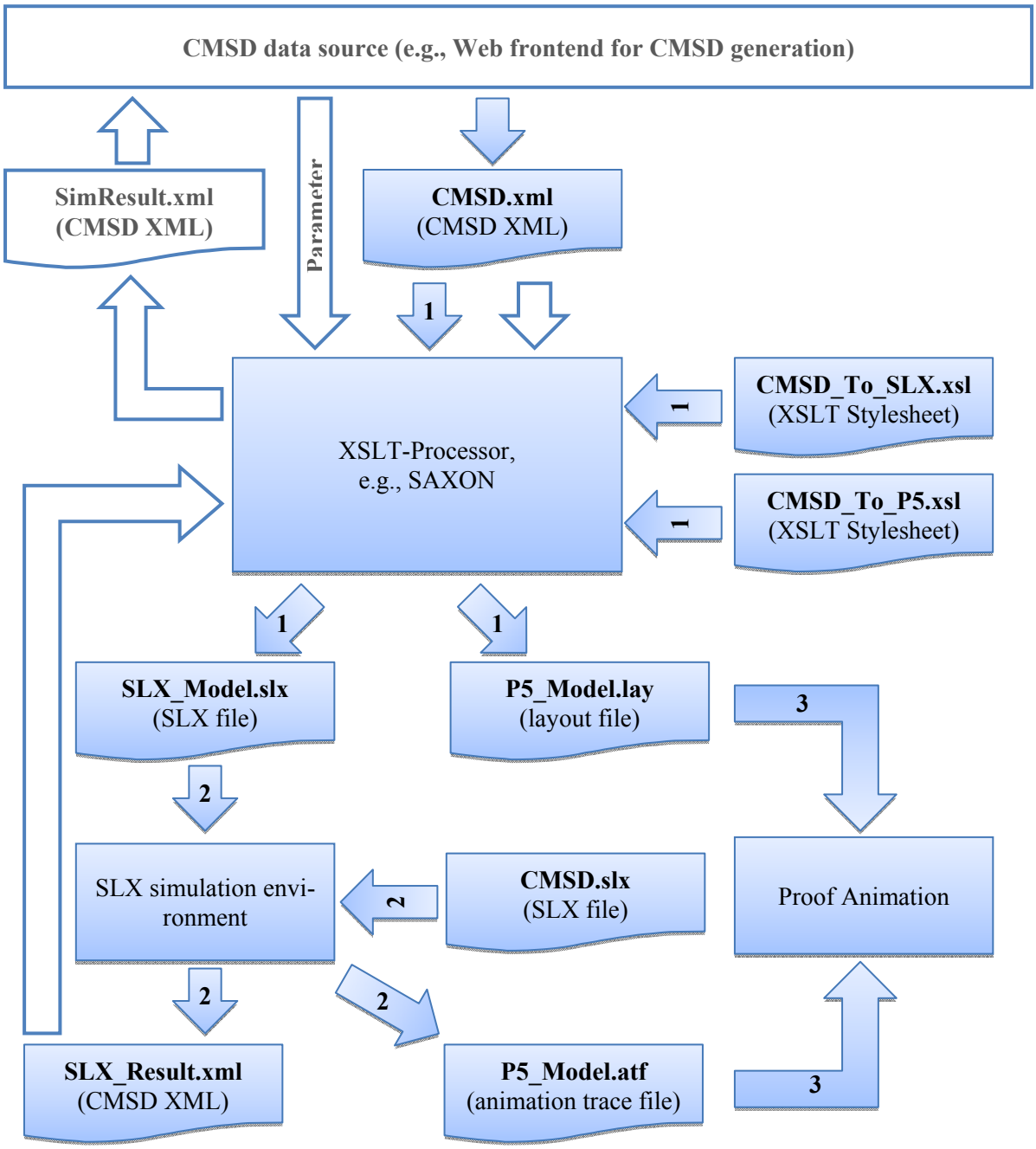

Figure 5: Schematic architecture and workflow of the model generator

At the end of the XSLT processing, two external functions are called by the XSLT processor with the purpose of executing both SLX and Proof Animations. These functions ("ExecuteSLXModell" and "ExecuteProofAnimation") are implemented in a JAVA library named SLXStarter.jar. These functions are executed sequentlially, i.e., first the simulator SLX is started and runs the created model (denoted as step 2 in Figure 5). In this step the predefined class library CMSD.slx is used by the generated SLX code. The run of the simulation model also automatically creates the trace file for Proof Animation. After the simulation run is completed, Proof Animation is started to visualize the simulation results (denoted as step 3 in Figure 5).

[CMSD_To_SLX.xsl]

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:java="java:SLX_Starter?path=jar:file:///C:…\SLX_Starter.jar!/">
…
<xsl:for-each select="CMSDDocument/DataSection/Resource">
   <xsl:variable name="ResourceType">
      <xsl:value-of select="ResourceType"/>
   </xsl:variable>
   <xsl:text>pointer(</xsl:text>
   <xsl:value-of select="$ResourceType"/>
   <xsl:text>) </xsl:text>
   <xsl:value-of select="Identifier"/>
   <xsl:text> = new </xsl:text>
   <xsl:value-of select="$ResourceType"/>
   <xsl:text>("</xsl:text>
   <xsl:value-of select="Identifier"/>
   <xsl:text>");</xsl:text>
   <xsl:value-of select="$newline"/>
   <xsl:value-of select="Identifier"/>
   <xsl:text>-&gt;m_Description = "</xsl:text>
   <xsl:value-of select="Description"/>
   <xsl:text>";</xsl:text>
…
   <xsl:value-of select="java:ExecuteSLXModell($SLXExecutionPath, … )"/>
…
```

[CMSD.slx]

```
…
public class Machine(string(*) identifier) subclass(Resource(identifier, machine){
   public double m_LocationX;
   public double m_LocationY;
   private pointer(Job) m_CurrentJob;
   …
   actions {
    pointer(Event) event;
    PA_Write "MachineIdentifier"(m_Identifier) m_Identifier;
    PA_Write "MachineStatus"(m_Identifier) m_CurrentStatus;
    while(TRUE) {
…
```

[SLX_Model.slx]

```
…
pointer(Machine) Ma3 = new Machine("Ma3");
Ma3->m_Description = "drilling";
Ma3->m_Name = "drill";
…
```

Figure 6: Excerpts from CMSD_To_SLX.xsl, CMSD.SLX, and a SLX_Model.slx file

To further illustrate the automatic model generation process, let us take a closer look at Figures 6 and 7. Figure 6 illustrates how a class (here: a resource of type machine) from a concrete CMSD.xml file (not part of Figure 6, but shown in Figure 4) is transformed into the appropriate SLX code. The upper part of Figure 6 shows a section from the XSL template describing the transformation rules (here: the transformation of a resource class) from CMSD into SLX code. The middle part shows the generic SLX implementation for resources of type machine. The lower part shows the generated SLX code which is the result of the XSLT transformation.

The execution of the generated SLX model produces two result files. SLX_Result.xml contains statistical key figures and trace data for further statistical analysis (not discussed in this paper). Second, an animation trace file for processing in Proof Animation named P5_Model.atf is created.

[CMSD_To_P5.xsl]

```
…
<xsl:call-template name="CreateLayoutObject">
    <xsl:with-param name="Identifier" select="$ResourceIdentifier"/>
    <xsl:with-param name="StartX" select="$LocationX"/>
    <xsl:with-param name="StartY" select="$LocationY"/>
    <xsl:with-param name="Type" select="string('Machine')"/>
</xsl:call-template>
…
<xsl:template name="CreateMachineClass">
    <xsl:text>Color L4</xsl:text><xsl:value-of select="$newline"/>
    <xsl:text>Define Class Machine</xsl:text><xsl:value-of select="$newline"/>
…
```

[P5_Model.lay]

```
…
Color L4
Define Class Machine
…
Fill 2.1516 -0.3169
Line -5 -5 -5 5
Line 5 -5 -5 -5
Line 5 5 5 -5
Line -5 5 5 5
…
Message MachineStatus 1 2 LJ -15 -13.5 BG Backdrop disassemble
Message MachineIdentifier 1 2 LJ -15 11.5 BG Backdrop Machine Name
…
CPO Buffer Ma3_E 298 350
CPO Machine Ma3 338 350
CPO Buffer Ma3_A 378 350
…
```

[P5_Model.atf]

```
…
write MachineIdentifier(Ma3) Ma3
write MachineStatus(Ma3) idle
…
place Part3 at 308.000 350.000
write MachineStatus(Ma3) busy
…
```

Figure 7: Excerpts from CMSD_To_P5.xsl, a P5_Modell.lay file and a P5_Model.atf file

Figure 7 illustrates the interplay between a concrete CMSD description of a system and a Proof Animation visualizing its simulation. The example is again based on the CMSD class from Figure 4. The upper part of Figure 7 shows the XSL transformation rule for transforming resources of type machine into Proof Layout objects. It should be noted that the position of layout objects must be contained in the CMSD source file. The creation of layouts for CMSD files not containing such information may be an area for future extensions. The middle part in Figure 7 shows the part of the automatically generated Proof layout file defining the machine layout class and placing it into the layout at the appropriate coordinates.

Also note that CMSD is not a graphic exchange format, i.e., it does not contain a detailed vector graphic representation of entities. We therefore assume and create a very simplistic graphic representation in Proof Animation automatically (shown in Figure 8).

Finally, the lower part of Figure 7 shows a part of the Proof Animation trace file concerned with state changes of the layout object. This file is produced during the execution of the previously generated SLX code. It uses the same naming conventions for layout classes and can therefore be directly used in Proof Animation for visualization of the simulation run.

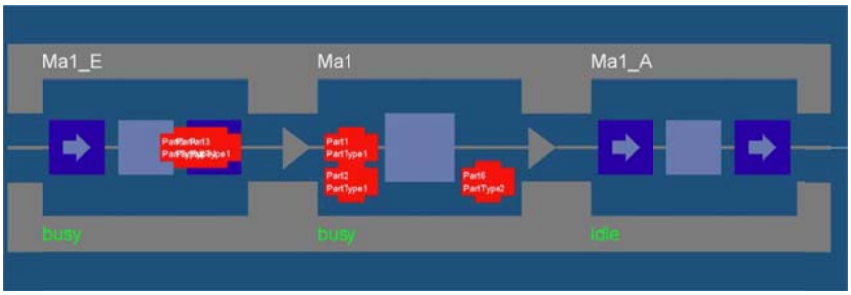

Figure 8: Screenshot showing a generated layout in Proof Animation. Here: a model of a machine with input and output buffers

The model generator for SLX has been tested with several scenarios implementing different job shop problems. Simulation results were further validated against results from a second CMSD-based model generator implemented in Plant Simulation (Bergmann, Fiedler, and Straßburger 2010). The execution of the models from both generators yields similar results. Both model generators offer similar functionality concerning the versatility of the supported modeling constructs. It should be noted that in the current state of development the model generators do not support all CMSD object classes. Some elements not typical to job shop problems have been omitted (e.g., conveyor systems). This is not a limitation of our conceptual reflections, but rather attributed to the complexity of CMSD. The implementation of additional model elements to both generators is certainly possible, but also strongly depends on how information in practical scenarios (e.g., data exported from an ERP system) is mapped onto the CMSD standard.

The advantage of the model generation approach presented in this paper is its flexibility and its degree of possible automation. All components can easily be encapsulated into web services and can form the technical basis for the integration of simulation as a plug-in into ERP or MES system architectures ("simulation as a service"). This also holds true concerning the extremely fast execution speed of the generated SLX models which clearly outperform equivalent models in Plant Simulation.

5 CONCLUSION AND FUTURE WORK

This article has introduced a new methodology for the automatic generation of simulation models in a language-based simulation environment. The methodology assumes a CMSD-based description of the simulated manufacturing system. We suggest the usage of XML stylesheet transformations to convert this CMSD model description into the actual source code of the chosen simulator.

The prerequisite for the suggested methodology is a generic implementation of the relevant CMSD classes and data types in the target simulator. In our prototype, this was easily achieved using the object-oriented features of SLX 2.0. The code generated by the XSLT transformation makes use of this simulator specific CMSD implementation, e.g., by creating instances of the generic classes and, in the case of SLX, activating their behavior. The article has further discussed the applicability of the methodology to the visualization system Proof Animation.

The advantage of the methodology is its flexibility and its suitability to work in web-based environments as it is based on well-established technologies from this sector. The suggested methodology is expected to be easily transferrable to other language-based simulation systems which contain basic features of object orientation.

Future work includes testing the methodology with larger manufacturing systems. Depending on these scenarios, additional functionality will be added to the model generator, e.g., for supporting workers and conveyors, as well as for extended collection of statistical data. Another venue of future work may include the validation of the methodology by adapting it to other simulation languages, like Desmo-J.

REFERENCES

Bergmann, S., Fiedler, A., 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 German)). In *Proceedings of the 14th ASIM Dedicated Conference on Simulation in Production and Logistics - integration aspects of simulation referring to equipment, organization and personne*l, Edited by G. Zülich and P. Stock, Karlsruhe Institute of Technology (KIT): 461-468.

Bergmann, S., Strassburger, S. 2010. “Challenges for the Automatic Generation of Simulation Models for Production Systems.” In *Proceedings of the 2010 Summer Simulation Multiconference*, July 11-14, 2010. Ottawa, Canada: 545-549.

Bergmann, S., Stelzer, S., Strassburger, S. 2011. ”Initialization of Simulation Models using CMSD”. In *Proceedings of the 2011 Winter Simulation Conference*, Edited by S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, December 11-14, 2011. Phoenix, AZ: 2228-2239.

Desmo-J 2012. ”*Desmo-J - A Framework for Discrete-Event Modelling and Simulation*”. University of Hamburg - Department of Computer Science. Accessed February 28, 2012. http://desmoj.sourceforge.net/home.html.

Fowler, J. W., Rose, O. 2004. “Grand Challenges in Modeling and Simulation of Complex Manufacturing Systems”. In *SIMULATION: The Society for Modeling and Simulation International*, 80(9): 469–476, September 2004.

Henriksen, J. O. 1997. “The Power and Performance of Proof Animation”. In *Proceedings of the 1997 Winter Simulation Conference*, Edited by S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, December 7-10, 1997. Atlanta, GA: 574-580.

Henriksen, J. O. 1999. “SLX - The X is for eXtensibility”. In *Proceedings of the 1999 Winter Simulation Conference*, Edited by P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, December 5-8, 1999. Phoenix, AZ: 167-175.

Henriksen, J.O. 2009. “Efficient Modeling of Delays in Discrete-Event Simulation”. In *Advancing the Frontiers of Simulation: A Festschrift in Honor of George Samuel Fishman*, Edited by C. Alexopoulos, D. Goldsman, and J. R. Wilson, 105-141. Springer 2009.

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*, Edited by S. G. Henderson, B. Biller, M.-H Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 1673-1681. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Schulze, T. and Henriksen J. O. 2002. ”*Simulation Needs SLX*”. Last modified April 2002. http://isgwww.cs.uni-magdeburg.de/pelo/sa/SimulationNeedsSLX.pdf.

SAXON 2012. *”SAXON - The XSLT and XQuery Processor”*. Accessed February 29, 2012. http://saxon.sourceforge.net/.

Siemens 2012. *“Plant Simulation*”. Product Lifecycle Management Software Inc. Accessed March 28, 2012. http://www.plm.automation.siemens.com/en_us/products/tecnomatix/plant_design/plant_simulation.shtml.

SISO 2010. “*Standard for: Core Manufacturing Simulation Data – UML Model*”. Core Manufacturing Simulation Data Product Development Group. Accessed February 14, 2012. http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=31457.

SISO 2012. “*Standard for: Core Manufacturing Simulation Data – XML Representation*”. Core Manufacturing Simulation Data Product Development Group. Accessed February 14, 2012. http://www.sisostds.org/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=32733&PortalId=0&TabId=105.

VDI 3633-1. “*Simulation of systems in materials handling, logistics and production - Fundamentals*”. VDI-Society Production and Logistics. Beuth Verlag, Berlin.

W3C (World Wide Web Consortium) 2006. "*Extensible Stylesheet Language (XSL) Version 1.1"* Last modified 05 December, 2006. http://www.w3.org/TR/xsl/.

W3C (World Wide Web Consortium) 2007. "*XSL Transformations (XSLT) Version 2.0*" Last modified 23 January, 2007. http://www.w3.org/TR/xslt20/.

W3C (World Wide Web Consortium) 2010. *"XML Path Language (XPath) 2.0 (Second Edition)"* Last modified 14 December, 2010. http://www.w3.org/TR/xpath20/4.

W3C (World Wide Web Consortium) 2012. "*Extensible Markup Language (XML)"* Accessed February 21, 2012. http://www.w3.org/XML/.

AUTHOR BIOGRAPHIES

SÖREN BERGMANN is a Ph.D. student at the Ilmenau University of Technology. He is a member of the scientific staff at the Department for Industrial Information Systems. He received his diploma degree in Information Systems from Ilmenau University of Technology. 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.

SÖREN STELZER is a Ph.D. student at the Ilmenau University of Technology. He is a member of the scientific staff at the Department for Industrial Information Systems. He received his diploma degree in Computer Science from the Ilmenau University of Technology. During his study he worked in the Neuroinformatics and Cognitive Robotics Lab of the Ilmenau University of Technology. After his study he worked in optimization of power plants in several projects. His research interests are simulation-based optimization, model predictive control, artificial learning and discrete-event simulation. His email is soeren.stelzer@tu-ilmenau.de.

SASCHA WÜSTEMANN holds a Bachelor degree in Information Systems from the Ilmenau University of Technology. He is currently completing his studies to achieve a Master degree at the Department of Industrial Information Systems of the same university. His interests include simulation of manufacturing systems and production control, especially using the simulation system SLX. His email is sascha.wuestemann@tu-ilmenau.de.

STEFFEN STRASSBURGER is a professor at the Ilmenau University of Technology in the School of Economic Sciences and is 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 Ph.D. and a Diploma degree in Computer Science from the University of Magdeburg, Germany. He is a member of the editorial board of the Journal of Simulation. His research interests include distributed simulation as well as general interoperability topics within the digital factory context. He is also the Vice Chair of SISO's COTS Simulation Package Interoperability Product Development Group. His web page can be found via www.tu-ilmenau.de/wi1. His email is steffen.strassburger@tu-ilmenau.de.