FLEXIBLE GENERATION OF REPORTS FOR SIMULATION-BASED EARLY WARNING SYSTEMS USING XML

Ingo Hotz
DaimlerChrysler AG
Gaggenau Plant, WG-PZP
Hauptstrasse 107, 76571 Gaggenau, GERMANY
E-mail: ingo.hotz@daimlerchrysler.com

Thomas Schulze
School of Computer Science
University of Magdeburg
Universitätsplatz 1, 39106 Magdeburg, GERMANY
E-mail: tom@iti.cs.uni-magdeburg.de

KEYWORDS

Online-Simulation, Simulation-based Early Warning Systems (SEWS), XML, XSLT, Flexible Reporting

ABSTRACT

Simulation-Based Early Warning Systems provide opportunities to control and analyze material flow systems by forecasting future system states. For this purpose a flexible reporting and extracting of simulation results respectively is essential but not supported by simulation tools in a standardized and simulator-independent way. There is a variety of general communication methods to exchange results with non-simulation applications (e.g. Simulation-Based Early Warning Systems). Nevertheless the data structure and representation have to be described for every isolated case.
This article presents a solution approach to enable flexible generation of reports via XML and XSLT. For this reason an XML schema definition used for defining variable calculation rules is described and its implementation in representative simulation tools is pointed up as well as the link to simulation-based early warning systems.

INTRODUCTION

Different customer requirements, the rapid change of available technologies and the bigger flexibility of production factors in the automotive industry lead to increasing competition and complexity in process and product innovation. Appropriate methods to cope with these challenges are efficient production strategies.
In this context, time is a very important factor. The sooner future developments are identified, the faster one can respond to them. Simulation-Based Early Warning Systems (SEWS) support a proactive control of real material flow systems. In consequence of real or potential state changes, proactive control (unlike reactive control) makes foresighted and target-oriented acting possible.
The usage of simulation as a part of an early warning system to control real material flow systems makes great demands on simulation models and the system environment. One requirement is that a potential user of SEWS does not have to parameterize, start and analyze simulation runs. For this reason the simulation model has to be embedded invisibly into a SEWS and special programming constructs are required that allow simulation models to be integrated in a production control or operating system (Banks 2000).
The main objective of this approach is to support SEWS with efficient functions to generate reports flexibly and independently of the used simulator. The following section will give a short description of SEWS and their architecture. After that the flexible generation of reports via variable definition will be discussed. The last section explains how simulation models can work with it. Finally the paper gives a conclusion and an outlook on future work.

SIMULATION-BASED EARLY WARNING SYSTEMS

The basic concept of SEWS is not new but not described conclusively in literature and consistently implemented in practice. One of the commonly used terms in this context is online-simulation, which is part of SEWS to forecast future system states. Generally online-simulation is the simulation of an existing real system based on a simulation model of this system. The simulation model has to be initialized with real data of the current system state. Results have to be calculated in an adequately short time (Schulze et al. 2003). There are a lot of terms and definitions which are related directly and indirectly with SEWS but it is possible to differentiate them (Hotz and Schulze 2006). The following section defines and explains the term of SEWS and introduces the system architecture.

Making SEWS Understandable

Generally early-warning systems are mechanisms deployed to inform persons at risk of imminent danger at an early stage. The purpose is to enable the user or the deployer of the early-warning system to prepare for the danger and act accordingly to mitigate against or avoid it (Greulich and Barnert 2003).
Simulation models are used to predict the behaviour of real and complex systems which are stochastically influenced. Typically simulation models help to analyze the effects of different action alternatives without implementation and negative effects for the real system.
SEWS combine the functionalities of simulation and early-warning systems. The simulation-based forecasting of future system states is the significant difference to classic early-warning systems which are based exclusively on historical and current measuring data.

Proceedings 20th European Conference on Modelling and Simulation
Wolfgang Borutzky, Alessandra Orsoni, Richard Zobel © ECMS, 2006
ISBN 0-9553018-0-7 / ISBN 0-9553018-1-5 (CD)

Recapitulating the following definition of SEWS may be useful:
"A Simulation-Based Early Warning System is a mechanism which is based on a simulation model of a complex real system and points negative effects or positive potentials out as soon as possible and informs the user of the complete system conveniently by forecasting and analyzing different action alternatives."
The potential users of SEWS are the responsible people who are commissioned to control the real system. In practice the acceptance of SEWS is affected by the capability of detecting exceptional circumstances, the generation of reasonable action alternatives and the flexible representation of important variables from simulation results.

System Architecture of SEWS

In principle the architecture of SEWS includes five primary components (Hotz and Schulze 2006):

- *Datasources:*

  Relevant system data has to be collected from different data sources. These could be databases of operating systems, production control centers or different production planning systems like enterprise resource planning or other planning and documentation databases. Maintenance data and data of financial controlling systems are also important. Specific scenarios necessitate collecting data from computer numeric controls, storage programmable logic controllers for conveyors or machines and OPC-server directly. There are a lot of approaches to connect data from real production facilities with simulation models (Feldmann 2000) and represent simulation relevant data in XML (Jensen and Reinhardt 2003).

- *Framework:*

  The main functions of the primary component *Framework* are data handling, the initialization and control of simulation models, the gathering and evaluation of simulation results. Furthermore it has to support an efficient communication with a SEWS-user and to generate appropriate action alternatives in the case of exceptions in the material flow system e.g. machine failures or material tailbacks.

- *Simulation Model(s):*

  The ability to forecast and proactive control is provided by simulation models of the real system. A SEWS should be independent of specific simulation models or simulators. Some simulators are more capable because of their performance, their communication abilities and their extensibility, others are not.

- *User and user interface respectively:*

  The control center user is of utmost importance. He is supplier of data as well as beneficiary of the system. Via a user interface he is able to configure the different functionalities of the framework depending on the real system.

- *Global XML-Structure and Web Services:*

  At the evolution of SEWS attention has to be paid to standardization, extensibility and reusability of the whole system and its components. XML is a standardized data exchange format and satisfies these requirements regarding extensibility and standardization. Web Services guarantee the reusability of commonly used functions which can be used by more then one SEWS. A Web Service is a service whose functionalities are described in XML and can be called over the internet and intranet respectively (Graham et al. 2002).

Figure 1 displays the system architecture of SEWS.

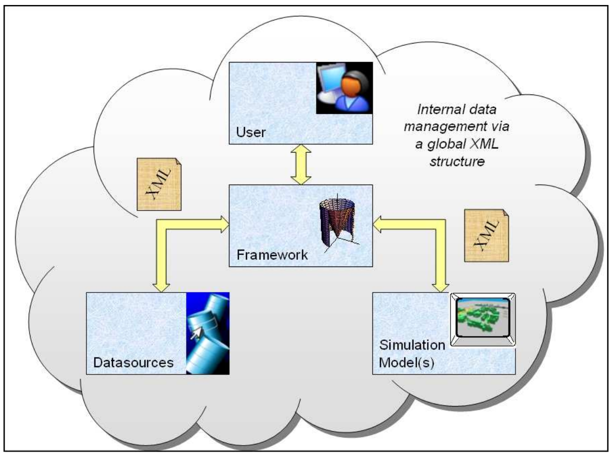


Figure 1: System Architecture of SEWS.

The following section describes the flexible report generation via variable definition.

FLEXIBLE REPORT GENERATION VIA VARIABLE DEFINITION

Using a variety of simulation tools causes the necessity of result format standardization. There are a lot of possible forms and methods to generate reports and results from simulation runs. If we want to apply a SEWS independently of the used simulator the report generation, the representation and the evaluation of simulation results have to be standardized.
Defining and parametrizing variables via XML are the method of solution in this article. XML can be used in different technologies and can be represented in a variety of display formats. For this reason it is necessary to describe the processing of XML documents, because a variety of illustration facilities for XML data is available e.g. web pages.
The variables are freely configurable by the control center user of a SEWS. He can flexibly define his own reporting as requested and thus independence can be achieved. The simulator and simulation model respectively can process the calculation rules of these vari-

ables and return the results after or during a simulation run.
First of all the following subsection illustrates the technology XML. Subsequently the definition of variables via XML and the generation of source code using XSLT will be described.

The World is Speaking XML

Recommended by the W3C (W3C 2006) the *eXtensible Markup Language* (XML) is a general-purpose markup language for creating special-purpose markup languages, capable of describing many different kinds of data. Derived from the *Standard Generalized Markup Language* (SGML), XML plays an important role in facilitating the sharing of data across different systems, especially systems connected via the Web.
Markup languages based on XML like MathML (MathML 2006), XHTML and XSLT are defined in a formal way and allow programs to modify and validate documents without prior knowledge of their form. Following points make XML well-suited for data exchange (McLaughlin 2002, Skulschus 2004, Bates 2003):

- Simultaneously human- and machine-readable. The support of unicode allows the communication of almost any information in any human language and it manifests as plain text files, independently of licenses or restrictions.
- The most general computer science data structures can be represented (records, lists and trees).
- A strict syntax and parsing requirements allow the parsing algorithms to remain simple, efficient and consistent.
- The self-documenting format describes structure and field names as well as specific values. A hierarchical structure suits to most types of documents.
- The robust, logically-verifiable format is based on international standards.
- Platform-independent and relatively immune to changes in technology.
- Extensive experience and software availability because XML and its predecessor SGML have been in use since 1986.

An XML document has to be well-formed and valid. Valid XML documents contain data that conforms to an XML schema that describes correct data values and locations. Schema definition formats for XML are the *Document Type Definition* (DTD) and the *XML Schema Definition* (XSD). XSD is the more powerful successor of DTD in describing XML languages because of its rich data typing system that allows more detailed constraints on an XML document's logical structure. Furthermore XSD is based on XML format and can be processed by ordinary XML tools.
Widely used *Application Programming Interfaces* (APIs) for processing XML data are the *Document Object Model* API (DOM) for random-access processing and the *Simple API for XML* (SAX) for serial processing. Data binding is another possibility of processing XML data and makes them available for programming language data structure. An example for data binding systems is the *Java Architecture for XML Binding* (JAXB).
There exist a lot of approaches to make the advantages of XML useful for simulation. One approach is to exploit the features of XML for modelling and simulation systems. Classes are generated which can be integrated in the target simulation system and support the user in constructing and editing models (Röhl and Uhrmacher 2005). Another related approach is to use XML-based code generators for converting simulation models between different run-time platforms without manual changes (Wiedemann 2005).
Usually XML is used for describing and exchanging data between different systems. Concerning a specific simulation problem, the relevant data are collected and represented in XML, so all applications involved in solving this problem work on the same data structure. Lee and Luo use XML to represent a machine shop data model and describe a mechanism for transferring data between a traditional database and XML files (Lee and Luo 2005). Further there are ambitions to describe simulation models formally in a universally valid way (Reinhardt et al. 2003).
The problem of any approaches is that there is no available standard XML schema definition for simulation models and it is utopian that this problem can be solved in the near future. The reason is complexity in simulation which makes it necessary to have simulation experts who are able to abstract problems from reality. To find a matching XML schema definition that supports every possible simulation problem under the restriction of necessary abstraction would be the solution of striking simplicity.
For this reason a global XML schema definition for SEWS is not designed to describe simulation models completely. It simply has to support the data management. The XML schema for describing variables which is defined in the following is only a fragment of the global XML schema definition for SEWS.

Defining Variables with XML

The precondition of flexible generation of reports is the presentability of variables. Simulation entities (e.g. loads, resources, workers, order lists, etc. …) own a lot of attributes depending on the used simulation tool. A flexible and open structure to describe different variable definitions and constellations is needed. Furthermore variables calculability consisting of several different parameters has to be supported.
An appropriate XML schema definition makes this approach possible. A variable is defined by recursive use of the XML node *operand*. An *operand* represents either an *element* or further *operations*. At this point the recursivity starts because an *operation* consists of two *operands* and one *operator*. An *element* includes the

following information about the related simulation entity (generally an entity is a representation of real things, Chen 1976):

- *ElementType*: Defines the type of simulation entity. Element Types are resources (e.g. machines), workers, loads, queues or order lists.
- *ElementID*: Represents the explicit identification of a specific simulation entity.
- *AttributeID*: Specifies the desired attribute. Its value has to be used for calculation.

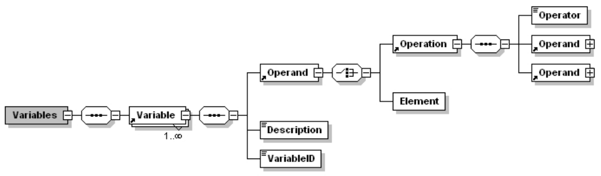


Figure 2: XML Schema Definition for Variables.

A variable calculation rule can be represented by a directed root tree (Neumann and Morlock 2002). Operands which contain an element are the leaves of this tree.
To clarify this coherences a simple example should be given. Let *a*, *b*, *c* and *d* be elements used to calculate the variable *x* as follows:

$$x = (a + b) * (c - d)\ .$$

This calculation can be displayed in a directed root tree (Figure 3).
Furthermore an element can contain an absolute term or whole calculation rules of other variables. For this reason very complex calculation rules are within the scope of possibility.

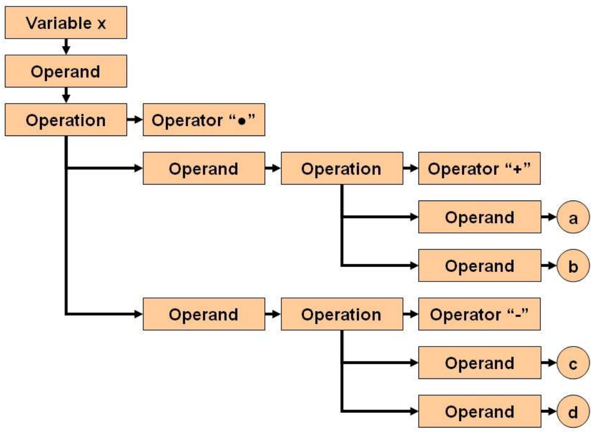


Figure 3: Root Tree of Variable *x*.

Defining variables in this way is not completely new. The *Mathematical Markup Language* (MathML) provides a similar systematic for representing mathematical expressions, symbols and formulae with the aim to integrate them into HTML documents. It deals with the presentation as well as information about the meaning of formula components.

At this point we have reached the first step of flexible generation of reports. We are able to define complex calculation rules of variables. Figure 4 illustrates an example XML document with two variables (*milling_output_1* and *worker_idle_state_1*) based on the following calculation rules:

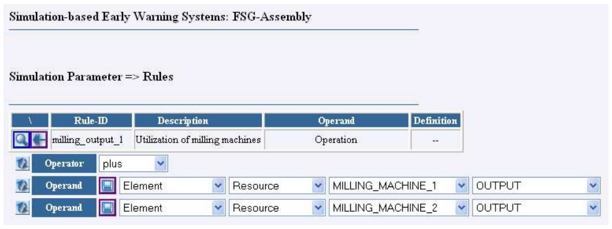
milling_output_1 =
  RESOURCE→MILLING_MACHINE_1→OUTPUT +
  RESOURCE→MILLING_MACHINE_2→OUTPUT

worker_idle_state_1 =
  WORKER→MILLING_WORKER_1→AVG_IDLE +
  WORKER→MILLING_WORKER_2→AVG_IDLE

```
<Variables>
    <Variable>
        <Operand operand_id="operand_1">
            <Operation>
                <Operator>plus</Operator>
                <Operand operand_id="operand_1_1">
                    <Element  attributeID="OUTPUT"
                              elementID="MILLING_MACHINE_1"
                              elementType="RESOURCE"/>
                </Operand>
                <Operand operand_id="operand_1_2">
                    <Element  attributeID="OUTPUT"
                              elementID="MILLING_MACHINE_2"
                              elementType="RESOURCE"/>
                </Operand>
            </Operation>
        </Operand>
        <Description>utilization of milling machines</Description>
        <VariableID>milling_output_1</VariableID>
    </Variable>
    <Variable>
        <Operand operand_id="operand_2">
            <Operation>
                <Operator>+</Operator>
                <Operand operand_id="operand_2_1">
                    <Element  attributeID="AVG_IDLE"
                              elementID="MILLING_WORKER_1"
                              elementType="WORKER"/>
                </Operand>
                <Operand operand_id="operand_2_2">
                    <Element  attributeID="AVG_IDLE"
                              elementID="MILLING_WORKER_2"
                              elementType="WORKER"/>
                </Operand>
            </Operation>
        </Operand>
        <Description>average idle state of workers</Description>
        <VariableID>worker_idle_state_1</VariableID>
    </Variable>
</Variables>
```

Figure 4: Example XML Document of Variables.

These variables are based on known attributes of simulation entities and can be built e.g. with the aid of web pages (Figure 5). Now these rules have to be made available for the simulation model.



Simulation-based Early Warning Systems: FSG-Assembly

Simulation Parameter => Rules

\	Rule-ID	Description	Operand	Definition
	milling_output_1	Utilization of milling machines	Operation	--

Operator	plus				
Operand		Element	Resource	MILLING_MACHINE_1	OUTPUT
Operand		Element	Resource	MILLING_MACHINE_2	OUTPUT

Figure 5: Defining Calculation Rules via Web Pages.

The idea to solve this problem is either to generate source code which is useable by the simulation model directly or to generate data files which can be read from the simulator and simulation models respectively. The best fitting alternative is depending on the used simulation tool. Anyway at this time XSLT comes into play.

Code Generation with XSLT

The *eXtensible Stylesheet Language* (XSL) describes how XML encoded files have to be formatted or transformed. XSL is used in three W3C-recommended language specifications (W3C 2006). These languages are *XSL Transformations* (XSLT), *XSL Formatting Objects* (XSL-FO) and *XML Path Language* (XPath).
XPath is used to address the parts of an XML document whereas XSL-FO specifies the visual formatting of XML documents.
Related to this papers topic the interesting specification is XSLT. It supports the transformation of XML documents into other document formats like HTML, RTF, TeX, plain text or even back into XML with new or changed values and attributes.
For the transformation two things are necessary. An XSL document describes the transformation output and an XSLT processor is doing the actual transformation. There are different XSLT processor distributions available. The very most common distribution is Xalan-Java and can be used from the command line, in an applet, a servlet or as a module in other programs.
Depending on the used simulator, XML documents can be transformed into different formats via XSLT stylesheets (Figure 6).

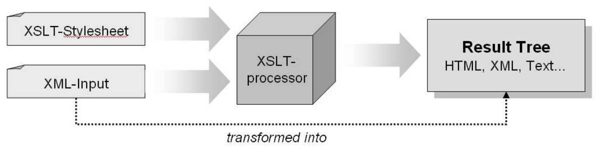


Figure 6: XSLT Transformation (Burke 2001).

Afterwards the simulator and simulation model respectively uses this data. At the end of simulation runs or at specific points of time the variables have to be calculated and returned. The processing of calculation rules is discussed in the following section.
The attributes of the XML node *element* is defined in a common way independently of the used simulator. It has to be translated into simulator specific attributes. XSLT stylesheets consist of template rules. During the XSLT processing the XML documents are processed according to a fixed algorithm to find XML elements and nodes respectively that meet certain conditions of template rules. Inside the template rules, instructions are processed like sequential instructions, which define the result document.
Then the generally defined attributes can be interpreted depending on the target simulator. Figure 7 demonstrates the stylesheet code for interpreting the *element* attributes of the example XML document in Figure 4 for the use in the Simulator "AutoMod".
The first template is invoked for every *element*-node in the XML document. The XML attributes related to this node (AttributeID and ElementType) are passed to a second template which is able to interpret the element type (in the example stylesheet: resource and worker). Depending on this element type a third template returns the simulator-specific attributes in cleartext.

```
<xsl:template match="Element" mode="auto_mod_attribute">
        <xsl:call-template name="auto_mod_attribute">
                <xsl:with-param name="attribute_id" select="@attributeID"/>
                <xsl:with-param name="element_type" select="@elementType"/>
        </xsl:call-template>
</xsl:template>

<xsl:template name="auto_mod_attribute">
        <xsl:param name="attribute_id"/>
        <xsl:param name="element_type"/>
        <xsl:if test="$element_type='RESOURCE'">
                <xsl:call-template name="amod_resource_attributes">
                        <xsl:with-param name="attribute_id" select="$attribute_id"/>
                </xsl:call-template>
        </xsl:if>
        <xsl:if test="$element_type='WORKER'">
                <xsl:call-template name="amod_worker_attributes">
                        <xsl:with-param name="attribute_id" select="$attribute_id"/>
                </xsl:call-template>
        </xsl:if>

        …

</xsl:template>

<xsl:template name="amod_resource_attributes">
        <xsl:param name="attribute_id"/>
        <xsl:if test="$attribute_id='OUTPUT'">
                <xsl:text>total loads</xsl:text>
        </xsl:if>
        <xsl:if test="$attribute_id='AVG_OUTPUT'">
                <xsl:text>average loads</xsl:text>
        </xsl:if>
        <xsl:if test="$attribute_id='DOWN_TIME'">
                <xsl:text>average down time</xsl:text>
        </xsl:if>

        …

</xsl:template>
```

Figure 7: Stylesheet for interpreting element attributes.

The following section describes the generated code and how it can be used by the simulation models and how the calculation rules are processed.

MAKE SIMULATION MODELS WORKING WITH IT

The objective target is to enable simulation models to calculate desired variables for a flexible reporting. At the moment one can produce *useful* output with the aid of XSLT and XML documents. Thus the next sections will describe how simulation models can be provided with this output, how simulation models process this data and how results can be calculated and returned.
First the communication method will be discussed. Afterwards the initialization of calculation rules in simulation models will be described. The explanation of the actual calculation of results and their representation will complete this section.

Simulation Model Communication

The described approaches in the preceding sections are tested with the simulation tools *AutoMod* and *SLX*.
AutoMod is a discret event-oriented simulator which is used in the area of material flow and logistics simulation. The modelling can be done in a graphical user interface. Complex control routines and workflows are developable using a specific simulation language.

This simulator provides the opportunity to communicate with other systems based on integrated interfaces (e.g. ActiveX, OLE, Client/Server) and to read data from files. Furthermore different graphical import formats like IGES, VRML and DXF are supported (Chung 2003).

The *Simulation Language with eXtensibilities* (SLX) consists of two components. On the one hand the SLX-Language which includes instructions to simulate discrete processes as well as general instructions like higher programming languages. On the other hand it contains general components to develop simulation systems. SLX is based on a C-like kernel which is augmented with basic functions for event-oriented discrete simulation which leads to the actual SLX kernel (Schulze and Henriksen 1998). For this reason almost all functionality of higher programming languages with their communication possibilities are available.

There are a variety of potential communication methods for generated source code. Usually company networks provide slow transfer rates and should not burden with network traffic unnecessarily. For this reason network traffic has to be hold minimal.

Making importable data files available to simulation models reduces communication costs to the required amount of data. This data files can be stored on the same computer the simulation model is running on. These files will be read or included at the beginning of simulation runs.

AutoMod simulation models as well as SLX simulation models have to be compiled. Compared to SLX the compilation process for bigger models developed in AutoMod is in the range of minutes. Thus data for simulation models are provided as readable data files. A model compilation is not necessary. Even the compilation of big SLX simulation models requires less then a minute. For this reason SLX code can be generated on the fly and imported as SLX classes. The code generation is one task of the SEWS framework which controls the data transfer and the simulation models. Figure 8 displays the flow of actions.

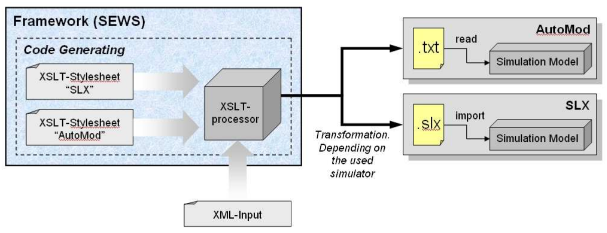


Figure 8: Code Generation depends on the Simulator.

To secure the existence of elements and their attributes, a validity check of calculation rules is inevitable in all cases.

The next question is how the application of calculation rules in AutoMod and SLX can be done. This will be discussed in the following section.

Initialization of Calculation Rules

The simulator and the simulation model respectively have to generate the desired variables at the end of simulation runs or at specific points of time. Prior to this the calculation rules have to be known. The generated code or data files may be used for the initialization of simulation models.

For the use in SLX there have to be defined classes which represent the structure of calculation rules analogical to the XML schema definition document. This structure can be allocated with data at the initialization. Figure 9 displays the module *variable_classes* which contains the classes to describe the XSD document explained in the section before.

```
//****************************************************************
// Module Initialization Classes of Calculation Rules
//****************************************************************
module variable_classes  {

    public class Element(string(*) elementID_){
        string(50) elementID;
        string(50) elementType;
        string(50) attributeID;
        initial{elementID = elementID_;  }
    }

    public class Operand(string(*) operandID_){
        string(50) operandID;
        pointer(Element) element_;
        pointer(Operation) operation_;
        initial{operandID = operandID_; }
    }

    public class Operation(){
        string(50) operator_;
        pointer(Operand) operand_1_;
        pointer(Operand) operand_2_;
    }

    public class Variable(string(*) variableID_){
        pointer(Operand) operand_;
        string(50) variableID;
        string(100) description;
        initial{variableID = variableID_;  }
    }

    public class Variables(){
        set(Variable) variable_;
    }
}
```

Figure 9: Initialization Classes in SLX.

The generated SLX code imports this module and instantiates objects which are described in the XML document using the defined classes. Figure 10 illustrates the generated SLX source code according to the example variable defined in Figure 4. First of all the elements with the identifier *MILLING_MACHINE_1* and identifier *MILLING_MACHINE_2* are created and assigned with its element type *RESOURCE* and its attribute *OUTPUT*.

Thereby *OUTPUT* is not a specific SLX attribute of a resource. It makes the connection between the generated source code and the example in Figure 4 easier.

The operands (*operand_1_1* and *operand_1_2*) are assigned to this element and it is integrated into one operation with the operator *plus*. This operation is a component of the first operand of the variable *milling_output_1* which is inserted into the list of variables. These defined objects can be used in the source code of the simulation model.

```
//*******************************************************************
// Module Initialization Values of Calculation Rules
//*******************************************************************
import "variable_classes.slx"

module variable_values{

    public class initialization(){
        pointer(Element) element_1_1_p;
        pointer(Element) element_1_2_p;

        pointer(Operand) operand_1_1_p;
        pointer(Operand) operand_1_2_p;

        pointer(Operation) operation_1_root_p = new Operation();
        pointer(Operand) operand_1_root_p;

        pointer(Variable) variable_1_p;
        pointer(Variables) variables_p = new Variables();

        initial{
            element_1_1_p = new Element("MILLING_MACHINE_1");
            element_1_1_p->elementType = "RESOURCE";
            element_1_1_p->attributeID = "OUTPUT";
            operand_1_1_p = new Operand("operand_1_1");
            operand_1_1_p->element_ = element_1_1_p;

            element_1_2_p = new Element("MILLING_MACHINE_2");
            element_1_2_p->elementType = "RESOURCE";
            element_1_2_p->attributeID = "OUTPUT";
            operand_1_2_p = new Operand("operand_1_2");
            operand_1_2_p->element_ = element_1_2_p;

            operation_1_root_p->operator_ = "plus";
            operation_1_root_p->operand_1_ = operand_1_1_p;
            operation_1_root_p->operand_2_ = operand_1_2_p;

            operand_1_root_p = new Operand("operand_1");
            operand_1_root_p->operation_ = operation_1_root_p;

            variable_1_p = new Variable("milling_output_1");
            variable_1_p->description = "utilization of milling machines";
            variable_1_p->operand_ = operand_1_root_p;

            place variable_1_p into variables_p->variable_; }
        }}
}
```

Figure 10: Generated SLX-Code for Instantiation.

Calculating and Returning Results

After providing generated source code and data files respectively to the simulation model, the structure and values of the calculation rules can be used.

```
begin F_calculateRule function
    set V_Operand_Type to F_GetOperand(Arg_OperandID)

    if (F_isOperation(V_Operand_Type) = true) then
    begin
        set V_OperationID to F_getOperation(Arg_OperandID)

        set V_Operator to F_getOperator(V_OperationID)

        set V_OperatorFunction to F_getOperatorFunction(V_Operator)

        set V_OperandID_1 to F_getOperandID(V_OperationID + "_1")
        set V_OperandID_2 to F_getOperandID(V_OperationID + "_2")

        set V_OperandValue_1 to F_calculateRule(V_OperandID_1)
        set V_OperandValue_2 to F_calculateRule(V_OperandID_2)

        set V_ReturnValue to
            F_calculateValue(V_Operator,V_OperandValue_1,V_OperandValue_2)

        return V_ReturnValue
    end
    else begin
        return F_getElementValue(V_OperandID)
    end
end
```

Figure 11: Calculating Rules in AutoMod.

The main target is to enable SEWS to generate flexible reports independently from the used simulator. At specific points of time the variables have to be calculated and returned. Such points of time could be the end of simulation runs or the emergence of heavy exceptions in the material flow system.

The recursive build and the tree structure of calculation rules prove an easy implementation of source code for calculating results in the simulation models. Elements represent the leaves of this tree structure and define the necessary attributes of simulation enitities. Their values are detected and used for calculation. Figure 11 illustrates an example source code in AutoMod.

Afterwards variables have to be returned in an adequate format. The simplest way is to return the pairing of *VariableID* and the corresponding calculated value. This can be done in an XML document which has to be generated from the simulation model. Figure 12 demonstrates a sample XML file which can be created easily with the output functions of simulators. This result file is readable and analyzable from the framework of the SEWS and other applications respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
<Results>
        <Result>
                <VariableID>milling_output_1</VariableID>
                <Value>232</Value>
        </Result>
        <Result>
                <VariableID>worker_idle_state_1</VariableID>
                <Value>0.13</Value>
        </Result>
</Results>
```

Figure 12: Result Representation via XML.

And now this article comes full circle. It starts with the defining of variables for flexible generation of reports and ends with the returning of results.

CONCLUSION AND FUTURE WORK

At the transmission plant Rastatt of the DaimlerChrysler AG several branches of production shall be provided with a SEWS as an experiment to verify the utilizability, benefits and potentials of SEWS in the automotive industry. The manufacturing process in transmission production basically consists of production areas, heat treating areas and assembling lines. Production areas manufacture important components like gears and shafts. These production areas are divided into machining before and after heat treating. The last step is the assembling of manufactured parts (Figure 13).

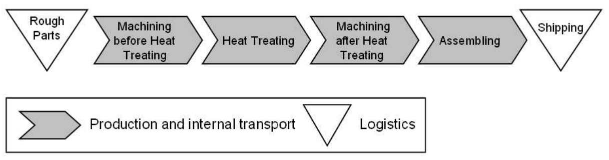

Rough Parts
Machining before Heat Treating
Heat Treating
Machining after Heat Treating
Assembling
Shipping
Production and internal transport
Logistics

Figure 13: Manufacturing Process in Transmission Production.

To test the functionalities of SEWS adequate representatives were chosen. These are the FSG assembly line which manufactures the transmission for the Mercedes-Benz A-class, the hardening stove A28 and the production control center of the shaft manufacturing for heavy duty transmissions.

Further developments to generate alternatives at the case of exceptions in material flow systems are necessary. SEWS have to be provided with a certain intelligence and learning aptitude. For this purpose an efficient systematics has to be located which is functioning independently of the supplied material flow system.

REFERENCES

Banks J. 2000. „Simulation in the Future“. In *Proceedings of the 2000 Winter Simulation Conference* (Orlando, FL, USA, Dec.10-13) IEEE, Picataway, N.J., 1568-1576.

Bates, C. 2003. “XML in theory and practice”. Wiley, Chichester.

Burke E.M. 2001. „Java and XSLT: Embedding XML processing into Java applications“. O’Reilly, Beijing, Köln.

Chen P.P.-S. 1976. “The Entity-Relationship Model – Toward a Unified View of Data”. In *ACM Transactions on Database Systems*, ACM-Press Vol. 1, No. 1 (March), 9-36.

Chung, C.A. 2003. “Simulation Modeling Handbook: A Practical Approach”. CRC Press LLC, Boca Raton.

Feldmann, K. 2000. “Simulationsbasierte Planungssysteme für Organisation und Produktion”. Springer, Berlin.

Graham, S.; Simeonov, S.; Boubez, T.; Davis, D.; Daniels, G.; Nakamura Y. and Neyama R. 2002. “Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI”. Sams Publishing, Indianapolis.

Greulich, W. and Barnert S. 2003. “Der Brockhaus Computer und Informationstechnologie”. *Der B-rockhaus*.

Hotz, I. and Schulze, T. 2006. „Simulationsbasierte Frühwarnsysteme – Definition, Anforderungen, Architektur“. In *Simulation und Visualisierung 2006*, T. Schulze, S. Schlechtweg, V. Hinz (Eds.). SCS European Publishing House, Erlangen, 63-77.

Jensen, S. and Reinhardt, A. 2003. „Integration industrieller DV-Systeme zur automatischen Modellgenerierung in der Getriebeproduktion“. In *Simulation und Visualisierung 2003*, T. Schulze, S. Schlechtweg and V. Hinz (Eds.). SCS European Publishing House, Erlangen.

Lee T.Y. and Luo Y. 2005. “Data Exchange for Machine Shop Simulation”. In *Proceedings of the 2005 Winter Simulation Conference* (Orlando, FL, USA, Dec.4-6) IEEE, Picataway, N.J., 1446-1452.

MathML World Wide Web. 2006. MathML (online). Available via *http://www.w3.org/TR/MathML2* (accessed January 29, 2006).

McLaughlin, B. 2002. “Java and XML data binding”. O’Reilly, Beijing, Cambridge, Farnham, Köln, Paris, Sebastopol, Taipei, Tokyo.

Neumann, K. and Morlock M. 2002. “Operations Research”. 2. Aufl. - München, Hanser, Wien.

Reinhardt, A.; Verzano, N. and Jensen, S. 2003. „Formale Beschreibung von Simulationsmodellen in XML“. In *Simulationstechnik – 17. Symposium in Magdeburg*, Hohmann, R. (Eds.). SCS European Publishing House, Erlangen, 69-74.

Röhl, M. and Uhrmacher A.M. 2005. “Flexible Integration of XML into Modeling and Simulation Systems”. In *Proceedings of the 2005 Winter Simulation Conference* (Orlando, FL, USA, Dec.4-6) IEEE, Picataway, N.J., 1813-1820.

Schulze, T.; Hanisch, A.; J. Tolujew, J. and Richter, K. 2003. „Online Simulation of Pedestrian Flow in Public Buildings“. In *Proceedings of the 2003 Winter Simulation Conference* (Louisiana, New Orleans, USA, Dec.7-10) IEEE, Picataway, N.J., 1635-1641.

Schulze, T. and Henriksen, J. 1998. „Simulation Needs SLX: Handbuch zum Simulationssystem SLX“. Otto-von-Guericke Universität Magdeburg, Fakultät für Informatik.

Skulschus, M. 2004. “XML Schema – Vollständige Einführung, Grundlagen, Praxis, Referenzen”. Galileo Press, Bonn.

W3C World Wide Web. 2006. XML (online). Available via *http://www.w3.org* (accessed January 29, 2006).

Wiedemann, T. 2005. “Simsolution – An Open Simulation Environment Founded on Extreme Multitasking”. In *Proceedings of the 2005 Winter Simulation Conference* (Orlando, FL, USA, Dec.4-6) IEEE, Picataway, N.J., 631-636.

AUTHOR BIOGRAPHIES

THOMAS SCHULZE is a Professor at the School of Computer Science at the Otto-von-Guericke-University, Magdeburg, Germany. He received the Ph.D. degree in civil engineering in 1979 and his habil. degree for computer science in 1991 from the University of Magdeburg. His research interests include modelling methodology, public systems modelling, manufacturing- and distributed-simulation with HLA. He is an active member in the ASIM, the German organization of simulation. His e-mail address is: `tom@iti.cs.uni-magdeburg.de`.

INGO HOTZ is working as a Ph.D. student in the production planning department of the transmission plant Rastatt, which belongs to the plant Gaggenau of the DaimlerChrysler AG. After the university degree in industrial engineering and management at the University of Karlsruhe, Germany in 2004 he is preparing his Ph.D. thesis at the University of Magdeburg. His main tasks are material flow simulation as well as developing digital planning methods. His e-mail address is: `ingo.hotz@daimlerchrysler.com`.