The 36th CIRP-International Seminar on Manufacturing Systems, 03-05 June 2003, Saarbruecken, Germany

Divide et Impera: A Flexible Integration of Layout Planning and Logistics Simulation through Data Change Propagation*

B. Mitschang[1], E. Westkämper[2], C. Constantinescu[1], U. Heinkel[1], B. Löffler[2], R. Rantzau[1], R. Winkler[2]

[1] Institute of Parallel and Distributed Systems

[2] Institute of Industrial Manufacturing and Management

University of Stuttgart, Germany

Abstract

The turbulent markets lead to new challenges for today's enterprises, they have to be transformable to stay competitive. Therefore, we developed a new approach that integrates Logistic Simulation and Layout Planning to fulfil the goal of improving the production system. Our approach is based on propagation and transformation of data changes concerning the continuous adaptation tasks among the Layout Planning and Logistics Simulation systems. Instead of relying on a tightly integrated global data schema, we connect systems only as far as required by building "bridges" between them. The systems that participate in the integration are kept autonomous. We use several state-of-the-art XML technologies in our integration system.

Keywords:

integration of information systems, XML technologies, facility layout planning, logistics simulation

1 MOTIVATION

Increased reliance on agile manufacturing techniques has created a demand for production systems to integrate the business processes organised along the value chain, leading to "islands of integration" within the enterprise. The actual level of integration within each island can significantly vary from one enterprise to another and important progress still remains to be made. This highly challenging process has recently evolved through several levels of integration from isolated legacy silos of enterprise information to agile and cooperative information systems, capable of effectively supporting the Agile Manufacturing Enterprise.

Technical challenges in supporting integrated layout planning and logistics simulation decisions in a complex and turbulent environment are manifold. The major challenges include: (1) the presence of multiple sources of uncertainty, both internal (e.g. machine breakdown) and external (e.g. new order arrivals, delay in tool production or raw material delivery), (2) the difficulty in accurately accounting for the finite capacity of a large number of resources operating according to complex constraints, and (3) the needs to take into account the multiple resource requirements of various operations (e.g. tools, raw materials, human operators).

From a process planning perspective, a considerable progress has been made by incorporating continuous adaptation of layout planning and manufacturing systems to new situations as mentioned above (e.g. machine breakdown, new order processing).

Simultaneously, important steps forward have been made in the development of agile and cooperative solutions for supporting the information systems integration. From an information system perspective, it is increasingly difficult to draw a line around an application system and then to own and control it. As the enterprise value chain extends along several business processes, multiple systems become part of each other's information systems. Furthermore, in many application areas, data is distributed over a multitude of heterogeneous, often autonomous information systems, and an exchange of data among them is difficult.

We present an innovative approach and a complex and detailed scenario of integrating the layout planning process and the logistics simulation process through a Data Change Propagation System. We suggest a flexible solution to this problem, based on the propagation and transformation of data concerning the continuous adaptation tasks among the layout planning and logistics simulation systems, using several state-of-the-art XML technologies.

We developed our layout planning/logistics simulation solutions and propagation system prototype as part of a larger research project on innovative concepts and techniques to enable highly flexible series production systems in the manufacturing industry.

The remaining paper is structured as follows. In Section 2 and 3 we introduce the Order Management Bench and the Facility Layout Planning System. Section 4 gives an overview of the Data Change Propagation System. A complex scenario of integrating the two systems is presented in Section 5. Section 6 outlines the future direction of our research and concludes the paper.

2 ORDER MANAGEMENT BENCH

The Order Management Bench — referred to as OMB hereafter — is a tool for designing, dimensioning and parameterising the complete order processing flow across distributed sites. The use of this tool allows users to model and analyse planning and manufacturing processes for production in variable environments. The OMB helps to analyse the dynamic interactions between planning, control methods, parameters and market behaviour on one hand, and manufacturing capacities as well as logistic efficiency of a production system on the other. The OMB is particularly useful in a turbulent

* This work is partially supported by the German Research Foundation (DFG/SFB 467)

environment where mean values are no longer applicable to production planning and control (PPC).

The detection of bottlenecks of different types is a central requirement when dealing with the analysis of as-is and to-be scenarios. In many cases, those bottlenecks affect the logistic goals only by means of a long event chain. Therefore, an approach is suggested that analyses the event chain on an event basis. It relates reason to impact and generates bottleneck indicators from that information. Those are called 'logistic opponents'.

The OMB enables the analysis of both the logistical interrelations and the impact of turbulence. Moreover, it can be used for the quick and reliable modelling, simulation, quantification, evaluation and implementation of the measures intended for goal-oriented change. The simulation model has a 'perfect' production data capturing system (PDC). This fact enables extended analysis based on event data that is usually unavailable without simulation.

2.1 Fields of Application

The OMB can be used in a variety of ways. Its main focus is on decision support and analysis of to-be scenarios during the design phase of a logistic system:

- As a design tool to change existing factories or to plan new ones: to design processes, capacities and order management strategies in consideration of cost and logistic efficiency.
- As a design tool to support the configuration of supply chains: to select production sites, and to define transportation capacities, disposition strategies, and production and warehousing capacities regarding the cost and efficiency of logistics.
- To dimension the customer order decoupling point: scenarios can be created to shift order decoupling points and parameters and highlight the resulting service quality. Thus, the interrelation between material planning type and/or parameter, inventory costs, guaranteed delivery date and schedule performance can be demonstrated.

2.2 Modules of the OMB

Modelling

The structural data and all parameters can be entered either directly into the database or via a specific graphical front-end. There are several versions of the 'modelling' module. On the one hand, the 'team based computer supported planning environment' [1] can be used as an input medium. On the other, a modelling environment was created employing the value stream symbols [2].

Simulation

The tool OMB is based on the paradigm of discrete event simulation. As simulation environment, eM-Plant (formerly known as 'Simple++', originally developed at the IPA [3]) is used. On top of that platform, a library of complex building blocks was developed. To give two examples, there is a building block 'PPC', that encapsulates all the backward scheduling mechanism and bill of material explosion used for MRP based planning. Another building block is the supplier, which encapsulates the delivery performance for different articles. There has been similar work done, but the special requirements for turbulent environments (e.g. capacity flexibility) have been omitted [4].

Model creation

During the modelling of a scenario, a descriptive, relational model is generated in the database. Based on the predefined simulation building blocks and during the automatic generation of the simulation model, the database information is used to create the complex structure of the simulation model. This approach was already applied in other environments [5][6]. The result is a model consisting of suppliers, producers, customers having on the upper level the material flow restrictions. The producer nodes consist of the configured planning level and an execution level comprising all work centres and stocks.

3 FACILITY LAYOUT PLANNING SYSTEM

Facility layout planning is usually the task of one or two experts in a company. Thus, the knowledge of the people from areas actually concerned is only insufficiently considered and their approval neglected. Small wonder that acceptance is low and the expertise of the shop floor workers remains by and large untapped. Often, the facility layout planning software (e.g. CAD applications) can only be used by experts. As a result, planning is first done on paper before being entered into the data processing environment, causing media discontinuities and bringing waiting time after it.

The team-based approach makes it possible to optimise factory planning and achieve staff participation, reduce waiting time and planning time, and simplify usage. The solution is to let all persons concerned jointly solve the planning task with the help of one software environment. This kind of support is provided by the planning table, which allows to visualise and document the planning process with 2D and 3D images, thus making it available for further internal data processing activities.

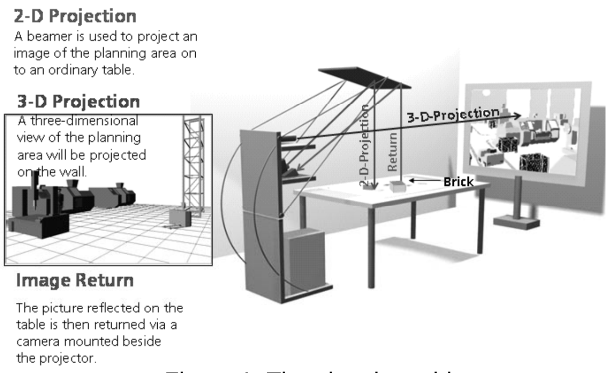
2-D Projection
A beamer is used to project an image of the planning area on to an ordinary table.
3-D Projection
A three-dimensional view of the planning area will be projected on the wall.
Image Return
The picture reflected on the table is then returned via a camera mounted beside the projector.
2-D-Projection
Return
3-D-Projection
Brick

Figure 1: The planning table.

The use of the planning table (Figure 1) reduces planning time and simplifies coordination between persons involved and concerned, avoids media discontinuities, cuts down the error rate, and eliminates any coordination effort in transferring planning results from paper to digital media.

The resulting layout – generated by the planning table – has to be evaluated against production data. Therefore, the Facility Layout Planning System (FLPS) uses a simulation tool. The material flow matrix acts as input for the simulation, which describes how much of a product in tons have been transported from one machine to another. This material flow data is calculated by the product data acquisition information contained in the product orders of the OMB. For this purpose an integration of both systems is necessary. Before the integration scenario is explained, the data integration system is presented that integrates both systems (OMB and FLPS).

4 THE DATA CHANGE PROPAGATION SYSTEM

4.1 Motivation for Data Change Propagation

The integration of the systems introduced above can be achieved by two approaches. The first one is to manage a central data store that feeds the two systems with their data. This approach is often infeasible or too expensive. The second one is to keep the autonomy of the systems by storing the data locally and managing the changes of overlapping data, so that the integrated systems can operate on the same data. Such an approach should be able to process the data changes through a data change mechanism. We call a system, which implements this flexible approach Data Change Propagation System [7][8][9].

The weakness of the first approach is the difficulty to agree on a common model for the integrated systems, derived from the heterogeneity of their IT infrastructures. For two systems it is still possible to find the common model, but in an enterprise with many applications and information systems it is difficult to find a global model. As described before, a transformable enterprise has to react quickly to turbulent market changes in order to attain the competitive advantage. These changes have an effect on the applications and information systems and their stored data. An adaptation to a global and rigid model is often difficult and costly.

The second approach, the integration of the systems by propagating the changed data, is appropriate for our purposes. The integrated systems remain autonomous ensuring that the systems are flexible enough to handle the required changes in a turbulent environment.

4.2 Overview of the Data Change Propagation System

The goal of the data change propagation process is to handle the data changes in one system and to forward the changed data to interested systems. A system is interested in a data change if it has to manage the same information locally. Due to system heterogeneity the data can vary in structure and the type of data storage. As we explain in the integration scenario in Section 5, the data can even have different granularity, e.g., one system stores the price of products and another system maintains the sum of these prices. The system where the data change occurred is called *source* system and the system, which is the recipient of a data change, is named *destination* system. A system can be a source and a destination at the same time. Based on the necessity to manage the same data, a dependency is defined as the “path of a propagation” from a source system to several destination systems.

There are three kinds of changes that can occur in the data: insert, update and delete. The destination systems have to be notified on the changed data and change type (insert, update and delete), so that the destination system can adapt their data accordingly.

4.3 System Architecture

Based on the approach presented above we developed a system called Data Change Propagation System. It consists of the following main components (Figure 2): the Repository, the Dependency Manager, the Propagation Manager, and the Queue Manager. The integrated systems are connected to the Propagation System by components called adaptors.

We envision now another component called Exploration Manager, subject for our future work, which analyses the propagated data by using business intelligence methods. The goal of such a component is to find chains of propagation, i.e., the fact that a propagation is followed by another one. These chains are the underlying business processes.

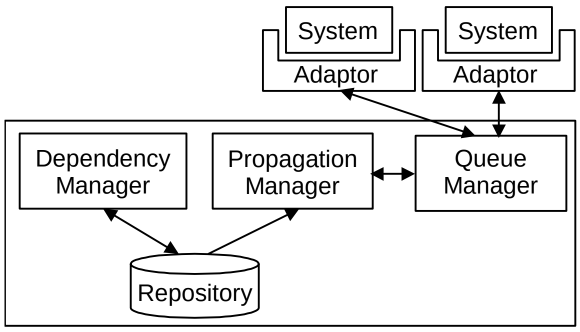

Figure 2: The architecture of the Data Change Propagation System.

The Repository

The Repository, the core component of the architecture, stores all information required for the data change propagation: the dependencies between systems, the associated propagation scripts and transformation scripts, the XML Schemas [10] of the systems and other relevant information about the connected systems.

The dependencies describe the paths of change propagation from a source system to multiple destination systems. The propagation scripts define the steps of processing the changed data: filtering, transforming and routing. These propagation scripts are explained in detail in Section 4.4. The transformation from the source data to the destination data is specified by the transformation scripts, described in Section 4.5. The structure of the propagated data is defined by the XML Schemas of the systems. The usage of XML Schemas allows the detection of errors in the transformation scripts and wrong data propagation from the source systems by checking the propagated data against these schemas. The system information contains basic data about the connected systems like the name and description of the system.

The Dependency Manager

The Dependency Manager is the design-time tool of the Data Change Propagation System. It handles the creation and updates of the information stored in the Repository. The Propagation Manager uses this information to propagate the changed data.

The Dependency Manager consists of two components, the Schema Editor and the Dependency Editor.

The Schema Editor manages the systems, the queues and the XML Schemas and stores them in the Repository. Like the Dependency Editor it has a graphical user interface that can be used to easily create all needed information.

The Dependency Editor is used to create and update all information related to a dependency. This information is the dependency itself, the propagation scripts and the associated transformation scripts. Additionally, it stores information about the connections among the scripts. This information must be kept consistent with the information stored in the scripts. Therefore, the Dependency Manager parses the edited scripts, extracts the information and writes this information in the tables of the repository.

The Dependency Manager is a tool that creates all scripts in text format. The propagation scripts can also be edited in a graphical notation. This notation is intuitive and easily to use and understand.

The Propagation Manager

The architecture of the Propagation Manager is illustrated in Figure 3. It consists of three main components: the Propagation Engine, the Filter and the Transformer.

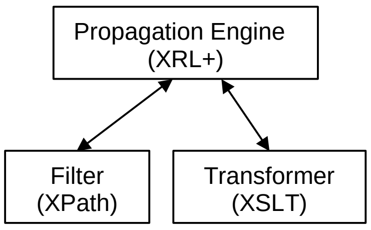

Figure 3: The architecture of the Propagation Manager. The terms in parenthesis show the technology employed.

The core component is the Propagation Engine, which executes the propagation scripts. Before these scripts are executed, a message from the source system has to be written into the input queue of the Queue Manager. The Propagation Manager reads the message and loads the appropriate dependency. This dependency is selected based on information from the message header regarding the source system and the schema. The associated propagation script is also read from the Repository. Then the Propagation Manager delivers the script to the Propagation Engine for execution. If the engine encounters instructions for filtering or transformation, then the Propagation Engine calls the Filter or Transformer. The transformation scripts are obtained from the Repository.

The Queue Manager

The Queue Manager manages three types of queues: the input queue, where the systems or the adaptors store the changed data, the output queues, where each system receives the changed data and processes it accordingly, and the internal queues, used to store intermediate results of the propagation process.

A queue is a data store used to exchange messages asynchronously, i.e., the receiver of such a message does not have to be available or waiting for a message when a message is sent. The message is stored persistently in the queue, with the advantage that a system crash can occur and the message is still available.

The Java Message Service (JMS) [11] implements the interface to these queues. For the implementation of JMS and the queues, the reference implementation from SUN Microsystems is used.

A JMS message contains three parts: the header, the properties and the body. The header consists of a predefined set of attributes, which describes the message. For instance, JMSDestination and JMSTimestamp are attributes that are defined in the header. The JMSDestination defines the name of the destination system and JMSTimestamp, the time when a message has been sent. Attributes that are important to describe a message, but are different from the header's default attributes are set in the properties part. An example of such an attribute is the type of change. As already mentioned, there are three different types of change: insert, update and delete of an object. The body of the message contains the data of the changed object. The contents of this message body is an eXtensible Markup Language (XML) document.

4.4 Propagation Scripts

The propagation scripts are created by using the eXchangeable Routing Language Plus (XRL+). This is an extension and adaptation of the workflow description language XRL [12][13], developed at the University of Eindhoven, according to the purposes of the data change propagation process. Before we describe the constructs of this language, we highlight the newly introduced elements:

- TRANSFORM (xml_in, xml_out, xslt),
- FILTER (xml_in, xml_out, xslt),
- MESSAGE_EVENT (system, schema, xml_out, change_type), and
- PROPAGATE (system, schema, xml, change_type)

The attributes of the elements above have the following meaning: *xml*, *xml_in* and *xml_out* refer to XML documents that are processed by the command. *Xslt* represents the name of the transformation script. *System* refers to the identifier of the source or destination system and *schema* to the corresponding identifier of the XML Schema, which describes the propagated data changes.

These new elements have the following meaning. TRANSFORM converts a source document into a destination document described by the transformation script. The FILTER instruction discards all XML documents that do not fulfil a specified Boolean expression. The Boolean expression is formulated in XPath [14], a language for accessing and evaluating XML documents. MESSAGE_EVENT represents an event, fired when a message from the specified system and schema has been written in the input queue of the propagation manager. Furthermore, an optional attribute for the type of change can be specified. This attribute indicates to wait for a message with a certain type of change (insert, update or delete). PROPAGATE writes the specified XML document into the output queue belonging to the destination system specified in the statement. The optional attribute change_type overwrites the change type. For example, this can be used to implement a propagation of bills of materials (BOM) from a Product Data Management System (PDM-System) to an ERP-System. The PDM-System stores BOMs through all steps of the product development life-cycle, while the ERP-System stores just the BOM involved in the production process. The update of a BOM in the PDM-System from the development state to production state will lead to an insert of this BOM in the ERP-System.

The XRL statements used for our approach are:

- waiting for events,
- timer events,
- parallel and sequential execution,
- conditional execution,
- loops, and
- termination of propagation processes.

The waiting instruction indicates to wait for one of two types of events: the message events and timer events. Message events have been presented above. The timer events represent events that are fired when the specified time interval has been finished or the specified point in time has been reached. The timer events make it possible to realise timeouts for message events. The instructions can be executed in sequential or parallel order. The execution of instructions can be associated with a condition that is realised as the implementation of the filter element using XPath. There are two branches of the conditional execution, one if the condition is true and one if false. There is also a while construct, which can be used to realise loops. This instruction is combined with a condition. If this expression is false the loop will stop the execution.

An example of a propagation script and its usage is given in Section 5.

4.5 The Transformation Scripts

The transformation scripts are implemented using the *eXtensible Stylesheet Language Transformations* (XSLT) [15]. XSLT is a language, which describes the transformation from XML documents to other XML documents or HTML documents. This is based on transformation rules specifying how an XML-Tag is transformed. Such an XSLT document contains a set of these rules and is processed by an XSLT-processor, e.g. Apache Xalan, which is used in our approach. The XSLT-processor tries to find a transformation rule for the root tag. The transformation rule can contain a command for selecting other rules. This makes it possible to process the XML tree representing the source document recursively.

An advantage of using XSLT is the wide availability of processors and tools like debuggers and graphical mapping tools. The debugger allows a step by step execution of a transformation script and the graphical mapping tool enables a user to create these scripts by drag and drop mechanism from the source schema to the destination schema.

XSLT can handle powerful transformations as is shown in the integration scenario. Additionally, the XSLT stylesheet can be extended with new functionality like Java methods invocations.

5 INTEGRATION SCENARIO

This section details the integration of our two systems of a transformable enterprise: the Order Management Bench (OMB) and the Facility Layout Planning System (FLPS). These two systems operate partly on the same data, but on a different granularity. First, we give an overview of the integration process. Second, we describe the processing of the production orders. Third, we give an example for a production order in XML and the resulting procedure call based on the Simple Object Access Protocol (SOAP) [16]. Finally, we present the used propagation and transformation scripts.

5.1 Overview of the Integration Scenario

There is a certain amount of information that can logically be shared between the Order Management Bench and the Facility Layout Planning System, e.g., information about the capacity of the resources needed in both systems. Having a closer look at the nature of the two problems brings up the differences: While layout planning works on “rough data”, the order management process needs much more detailed information. While the layout planning regards capacity of a machine as a static mean value that is used to calculate the overall output of a machine, the Order Management Bench has to deal with capacity flexibility and therefore needs information about different possible shift models.

As a general effect, the data processed by both tasks is very similar, but has differences regarding the granularity.

Since both tools are tailored to their problem domain, the underlying data models are different. For an online exchange of data, a transformation and adaptation of the changed data is needed.

Vital information for both systems represents the material flow: While layout planning usually needs information about the material flow between machines in terms of e.g. tons per day, represented as a material flow matrix, order management bench requires much more in depth analysis of the structure of the material flow and therefore handles individual work orders of the production site.

The situation becomes more complex when the tools (OMB and FLPS) need to have an online connection to reflect the results. We assume, that the OMB is used to simulate the basic decisions of layout planning and order management in an abstract way, without visualisation. During the simulation, the dynamic behaviour has to be presented in the visual environment of the FLP system. For this purpose, the volume or weight of individual orders in the OMB have to be aggregated for a short interval.

This example is used to give an impression of the problem domain and how our Data Change Propagation System solves this problem.

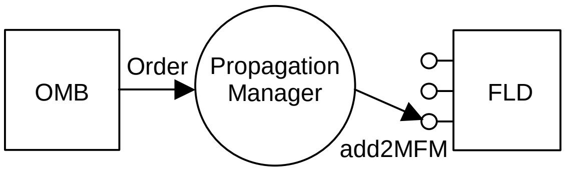

Figure 4: The general flow of an order from the OMB to the FLP system.

Figure 4 illustrates the flow of a production order from the Order Management Bench to the Facility Layout Planning System. A new order arrives and the OMB processes it. An order contains information about the processing steps and the associated machines. The Propagation Manager uses this information to provide an appropriate material flow matrix. The Propagation Manager calls a function of the interface implemented by Facility Layout Planning System. The pure data from the production order is transformed into the call of the function add2MFM (add to material flow matrix) with a material flow list as its argument. The FLP system adds this new information about the material flow of the new order to the material flow of all orders. Additionally, the Facility Layout Planning System offers functionality to initialise and clear the material flow matrix.

The following sections detail the scenario of the production order transformation.

5.2 The processing of production orders

Before the processing of production orders starts, the FLP system has to initialise the material flow matrix. This is done by the call of the init function of the interface, offered by the FLP system. This function clears all entries in the material flow matrix. This function call is handed over to the FLP system without any further processing in the Propagation Manager.

While the condition of processing production orders is true, a loop is executed. The first task in this loop is to send a reset event to the FLP system. This event is also sent to the FLP system without any further processing. The second task is to execute another loop for a certain time period. This period was specified by the FLP system in an initial message to the OMB. Thus, the FLP system can specify the time interval for the material flow matrix calculation.

As a first step of this loop the actual production order is sent to the FLP system. But the destination system does not know what to do with the production orders. Therefore further processing is needed in the Propagation Manager. First, it filters out all orders that are not needed for the calculation of the material flow matrix. These are represented by orders with low priority. Afterwards, the Propagation Manager calculates a material flow list (MFL) by using the transformation script “Order2MaterialFlowList”. We use a material flow list instead of a material flow matrix, because matrices cannot be encoded directly in an XML document. These data have to be added to the temporal matrix stored in the FLP system. There is a need for a procedure call that implements the adding, but the calculated MFL represents pure data, so the Propagation Manager has to transform this data into a procedure call. This is done by a second transformation script. The result is forwarded to the FLP system. The FLP system uses the propagated information to add the transport information to the local stored material flow matrix. The loop is repeated as long as orders arrive for the specified time span.

The algorithm for calculating the material flow matrix in pseudo code is sketched in Figure 5.

```
CalculateMaterialFlowMatrix()
  Send InitProcessing();
  While(ProcessingProductionOrders)
      Send Reset_Event;
      While(ProcessingInATimeWindow)
          Send Order;
          Filter low priority orders;
          Transform Order2MaterialFlowList;
          Transform to FunctionCall;
          Send to FLP-System;
      End While
End While
```

Figure 5: The calculation of the material flow matrix.

5.3 From Data Propagation to Procedure Calls

As mentioned before, the Order Management Bench (OMB) needs to call a procedure like add2MFM, which is not called directly by the OMB. Instead the OMB sends new production orders to the Propagation Manager, which transforms it into a procedure call.

Since the Propagation Manager uses XML technologies for its implementation, we decided to employ further XML technologies for calling procedures in the Facility Layout Planning System. There are two specifications for executing remote procedure calls (RPC) that are using XML for procedure marshalling. These technologies are the Simple Object Access Protocol (SOAP) [16] and XML-RPC [17].

XML-RPC is a lightweight protocol for calling RPCs over HTTP. It contains the name of the called procedure and a list of parameters, which are just specified by a data type and a value. The names of the parameters are not specified. The parameters can be of a scalar type or structures. Additionally, the specification defines how a procedure returns values that are written in a HTTP-Response and how errors are handled to the client.

A more complex protocol for calling RPC encoded in XML is SOAP. The advantage of SOAP over XML-RPC is that SOAP is not fixed to one protocol. It can be used together with HTTP or other protocols and transport mechanism. So the SOAP RPC can be used in conjunction with our transportation system, i.e., the message queues.

Another possible way is to develop an own approach for encoding an RPC. This can be done for example by using the header and properties fields of the propagated messages. Such a field can be the ProcedureName and a field for highlighting if the transported message is a RPC. This approach has two significant disadvantages: first, our transformation approach cannot be used to transform from data to a procedure call by the usage of transformation scripts, and second, there is no tool support available. These are the main reasons based on which we are using a standard for encoding RPCs. As we show later, the transformation scripts can be used to transform from data to RPCs encoded in SOAP, but it is also possible to have XML-RPC as target encoding style.

As already mentioned we can use SOAP together with our message queues. This is not possible for XML-RPC. A further weakness of using XML-RPC is represented by the use for parameters descriptions only the data types and their corresponding values. The missing parameter names induce here ambiguity on the semantic. Additionally, SOAP offers advanced features, like support for transactions and client state. An advantage of XML-RPC is the simple encoding of RPCs. Based on the arguments presented above we decided to use SOAP for RPC encoding.

5.4 Example for a Production Order and the Resulting Procedure Call

```
<ProductionOrder>
    <Planned_Due>11.02.2002</Planned_Due>
    <Planned_Start>09.02.2002</Planned_Start>
    …
    <Product> … <Weight>20 kg</Weight></Product>
    <Amount>200</Amount>
    <BillOfMaterial>... </BillOfMaterial>
    <Routing>
        <Operation seqNo=”01” Resource=”M001”
            start=”09.02.2002” finish=”09.02.2002” … />
        <Operation seqNo=”02” Resource=”M005”
            start=”10.02.2002” finish=”11.02.2002” … />
        <Operation seqNo=”03” Resource=”M007”
            start=”11.02.2002” finish=”11.02.2002” … />
        …
    </Routing>
</ProductionOrder>
```

Figure 6: An example of a production order.

Figure 6 illustrates an example of a production order that is sent from the Order Management Bench to the Propagation Manager. At the Propagation Manager the production order is transformed into a MaterialFlowList. This list is used by the Facility Layout Planning System to calculate the material flow matrix. For this calculation of transportations the sequence of operations is analysed.

Figure 7 illustrates the encoding of a procedure call for the procedure “Add2MaterialFlowMatrix” using SOAP. It consists of three parts: the envelope, the header and the body. The envelope contains the two other parts. The envelope root tag defines the SOAP namespace (xmlns:soap=”…”), used to group element tags that belong to each other. Additionally, the envelope defines the type of encoding.

The header block of the SOAP document can contain information about transactions, client state and more. This extra information is not needed in our scenario and since the header block is optional, we omit it here.

The body contains the payload of the transported RPC. The RPC is encoded by the name of the procedure (“Add2MaterialFlowMatrix”) and the arguments. Here we have one argument “MaterialFlowList”. This list is calculated by a transformation from the corresponding production order and consists of multiple entries with transport information. The transport information represents the source and destination of a transportation step, the transported part and the amount of transported parts.

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap=
        "http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle=
        "http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
    <Add2MaterialFlowMatrix>
        <MaterialFlowList>
            <MFLEntry start=”M001”
                destination=”M005””
                amount=”4000 kg”/>
            <MFLEntry start=”M005”
                destination=”M007”
                amount=”4000 kg”/>
            . . .
        </MaterialFlowList>
    </Add2MaterialFlowMatrix>
</soap:Body>
</soap:Envelope>
```

Figure 7: Resulting procedure call.

5.5 The Propagation and Transformation Scripts

Figure 8 illustrates the propagation script in a graphical notation. The propagation script implements a part of the algorithm that has been introduced above. It implements the tasks done by the Propagation Manager.

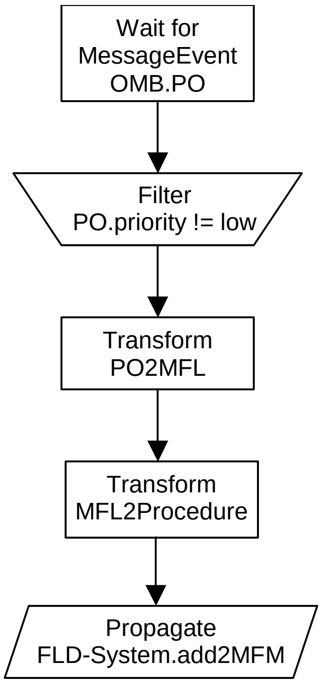
Wait for MessageEvent OMB.PO
Filter PO.priority != low
Transform PO2MFL
Transform MFL2Procedure
Propagate FLD-System.add2MFM

Figure 8: The propagation script.

The first statement in this propagation script is a wait for a message arrival in the input queue from the OMB system. It contains the type of data representing a production order (PO). If such a message has been put in the input queue, it will be read and the filter statement will be processed on this input data. The input data is discarded when the priority of the production order is low. This statement is followed by a transformation that is used to calculate the material flow list. The transformation is followed by another one, which transforms the pure data in a procedure call using the SOAP mechanism. This step encodes the procedure call in an XML format. Finally, the propagation statement writes the SOAP document into the output queue of the FLP system.

The simplified transformation script for the calculation is presented in Figure 9. This script is formulated in a pseudo code for ease of presentation. There is one transformation rule defined for the element “ProductionOrder”. In this rule, the output element “MaterialFlowList” is generated. Then the total weight is calculated from the amount of products that has been produced and the weight of the product. This information is used as a hint for the transportation weight. After this calculation a loop is executed over all operations starting from the second operation. Then the transport path is calculated by getting the previous machine (resource) and the current resource. For this transformation script we assume that the operations of the production order are in sequence order. If not the operations have to be sorted by the sequence number.

```
TransformationRule match=”ProductionOrder”
    <MaterialFlowList>
    TotalWeight= Amount*”Product/Weight”
    Foreach “Routing/Operation” StartPosition=2
        <MFLEntry start=”./preceding/Resource”
            destination=”./Resource”
            amount=”$TotalWeight”/>
    End Foreach
    </MaterialFlowList>
End TransformationRule
```

Figure 9: The transformation script PO2MFL.

The transformation script MFL2Procedure (Figure 10) is composed of one transformation rule that handles the addition of SOAP specific elements. In our example, this consists of the procedure name (Add2MaterialFlowMatrix). Inside the procedure name the argument *MaterialFlowList* is obtained, respectively copied, from the source document.

```
TransformationRule match=”MaterialFlowList”
    <soap:Envelope
        xmlns:soap=
          "http://www.w3.org/2001/12/soap-envelope"
        soap:encodingStyle=
          "http://www.w3.org/2001/12/soap-encoding">
      <soap:Body>
        <Add2MaterialFlowMatrix>
            CopyFromSource “MaterialFlowList”
        </Add2MaterialFlowMatrix>
      </soap:Body>
    </soap:Envelope>
End TransformationRule
```

Figure 10: Transformation script MFL2Procedure.

6 CONCLUSION AND FUTURE WORK

Based on the new agile manufacturing techniques, the production systems need a flexible, loosely coupled approach to propagate data changes between information systems while preserving their data management autonomy. We suggest a software component, called Data Change Propagation System, that manages dependencies between data stored in potentially different schemas, but only the data schemas of the systems connected to the Propagation Manager are stored in a repository. The Propagation Manager transforms an XML input message into an XML output message based on a transformation specification that has been defined for a data dependency. Such a transformation can be composed of several smaller transformation activities (XSLT scripts). By employing script templates, our approach allows for a reuse of transformation code for many dependencies.

In some situations it is useful to prevent data propagation. This can be achieved by applying constraints (XPath expressions) on the message input or on the intermediate results of transformations.

Furthermore, we developed a way of transforming propagated data into procedure calls by employing the Simple Object Access Protocol (SOAP). This is used to integrate to production systems: the Order Management Bench and the Facility Layout Planning System. These systems are integrated by propagating and transforming production orders, which are used to calculate a material flow matrix.

Our current propagation approach and system implementation manages one-to-one and one-to-many data dependencies. We currently investigate how to manage many-to-many dependencies using a single XRL script. This may have positive performance effects for the Propagation Manager because only one script instead of several would have to be processed. We intend to employ the XQuery language to query the Repository. Furthermore, we plan to integrate XQuery statements into the transformation scripts to offer even more flexibility to the transformation process. This would add another XML technology to our system, thus making it an XML-dominated middleware component. As already mentioned we are going to design a further component of the Data Change Propagation System, called Exploration Manager, which analyses the propagation processes using business intelligence techniques.

7 REFERENCES

[1] Sihn, W., von Briel, R., 2000, Interactive Factory Planning, International Conference on Manufacturing Engineering ICM.

[2] Mittelhuber, B., 2002, Simulationsbasiertes Wertstromdesign, Industrie-Management, 2002/1: 44-47.

[3] Becker, B., 1991, Gegenstandsorientiertes Simulationssystem mit parametrisierter Netzwerkmodellierung für Fertigungsprozesse mit Stückgutcharakter, IPA – IAO Forschung und Praxis (PhD Thesis).

[4] Scholtissek, P., Glaessner, J., 1997, Exploiting logistic potentials with a simulation-aided test of PPC methods, Production Planning and Control, Taylor & Francis, 8/1: 56-61.

[5] Biethahn et. al, 1999, Simulation als betriebliche Entscheidungshilfe, Physica-Verlag.

[6] Richter, H., März, L., 2001, Generic simulation: Creating the model's structure from a database, Proc. 20th IASTED Conference Modelling, Identification and Control, 740-744.

[7] Constantinescu, C., Heinkel, U., Rantzau, R., Mitschang, B., 2002, A System for Data Change Propagation in Heterogeneous Information Systems, Proc. 4th International Conference on Enterprise Information Systems (ICEIS), 73-80

[8] Constantinescu, C., Heinkel, U., Meinecke, H., 2002, A Data Change Propagation System for Enterprise Application Integration, Proc. 2nd International Conference on Information Systems and Engneering (ISE), 129-134.

[9] Rantzau, R., Constantinescu, C., Heinkel, U., Meinecke, H., 2002, Champagne: Data Change Propagation for Heterogeneous Information Systems, Proc. of the International Conference on Very Large Databases (VLDB), Demonstration Paper, 1099-1102.

[10] World Wide Web Consortium, 2001, XML Schema Part 0: Primer, available at http://www.w3c.org/TR/xmlschema-0

[11] Hapner, M., Burridge, R., Sharma, R., Fialli, J., Stout, K., 2002, Java Message Service – Version 1.1, Sun Microsystems.

[12] van der Aalst, W., Kumar, A., 2000, XML Based Schema Definition for Support of Inter-organizational Workflow. Proc. of 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000).

[13] Kumar, A., Zhao, Z., 1998, Workflow Support for Electronic Commerce Applications. Proc. of International Conference on Telecommunications and Electronic Commerce.

[14] World Wide Web Consortium, 1999, XML Path Language (XPath) – Version 1.0, available at: http://www.w3c.org/TR/xpath.

[15] World Wide Web Consortium, 1999, XSL Transformations – Version 1.0, available at: http://www.w3c.org/TR/xslt.

[16] Scribner, K., Stiver, M., 2000, Understanding SOAP – The Authoritative Solution, Sams Publishing.

[17] Winer, D.: XML-RPC Specification, available at: http://www.xml-rpc.org/spec