Automatic Transformation of Hospital Processes into an Executable Model with EPML

Matthias Kühn
Technical University Ilmenau
Ehrenbergstraße 29
98693 Ilmenau, Germany

Joachim Lippold
Technical University Ilmenau
Ehrenbergstraße 29
98693 Ilmenau, Germany

Horst Salzwedel
MLDesign Technologies Inc.
2230 Saint Francis Drive
CA 94303 Palo Alto, USA

ABSTRACT
Executable specifications are realized by models in simulation software systems. This requires collection of all necessary information about the real system and its environment. It is critical that all relevant information is included in the model and the correct level of abstraction is being used.

Often, process descriptions are available for real systems (e.g. hospitals). To create executable specification out of existing documentations a transformation is required. The process descriptions are generally incomplete, often not formal standardized and thereby not able to be transformed into a simulation model. Therefore the appropriate process description language has to be selected and generally extended (e.g. to include resources) to enable an automated transformation into a simulation model.

This paper describes how hospital processes (e.g. clinical pathways) can be transformed automatically into a simulation model using extended event driven process chains (eEPC). The event driven markup language (EPML) is used as interchange format. The transformation rules are defined by extensible style sheet language transformations (XSLT).

KEYWORDS
Process transformation, process modeling, simulation, hospital, EPML.

1. INTRODUCTION
Creating a comprehensive unified simulation model (executable model) of a dynamic and complex process such as the operational workflow of a hospital has shown to have a dramatic impact on reducing operating costs through process optimization. Traditional process improvements have been largely based on analytics and statistics coupled to experience and relative industry “role models”. These methods cannot analyze operational feedback to optimize the process and cannot cope with the non-linearities of those processes. Thus, advanced simulation methods to augment traditional analytics, becomes more and more important in development and evaluation of such complex projects. Faults and shortcomings, which are identified in early design stages of a project, can be removed with fewer costs and less time consumption than in later project stages [1, p. 19]. Davis [2, pp. 25 ff.] mentioned a multiplication factor of 10 of the relative costs for fixing specification failures during implementation phase. For that reason it is recommended to work with executable specifications (dynamic models), which are able to represent dynamic processes and allow validation and verification of specifications [3]. Thus, it is possible to overcome failure in early design stages, also in hospital projects as the increasing number of simulation studies shows [4, 5, 6]. Using executable specifications, improvements and potentials can be identified; different solutions can be tested and validated before implementation. Unintended interruption of the operation within the hospital as well as any endangerment of patients’ health can be avoided as shown in [6].

Hospitals and clinical institutions rarely have the necessary expertise to develop a simulation model on their own and have to establish necessary knowledge for a long term or need to use expensive external sources.

Current challenges are seen in facilitating model building and system simulation for hospitals and clinical institutions. Therefore in [7] a library of standardized building blocks was developed and validated for modeling, simulation, and optimization of hospital processes. These building blocks can be used for an easy and fast building up of a simulation model within a simulation system environment.

In this paper it will be shown how hospital processes and clinical pathways can be transformed into an executable specification. Clinical pathways and process documentations (e.g. for ISO 9001 certification) are common in hospitals. If an executable specification is built up out of these process documentations, it will guide to an easier and faster way of building up simulation models and facilitate simulation model building for hospitals and clinical institutions.

For process documentation a standardized format is necessary. Within this paper and further work, extended event driven process chains (eEPC) are used. The hospital process used here consists of 15 clinical pathways of a cardiologic department of a university hospital. The mentioned clinical pathways include hospital admission, anamnesis by nurses and physicians, examination by physician, blood analyses, electrocardiographic and echocardiographic examinations, treatment in a heart catheter laboratory and discharge of patients. Handling of emergency patients and outpatients is also included.

The transformation process described is visualized in the following Figure 1.

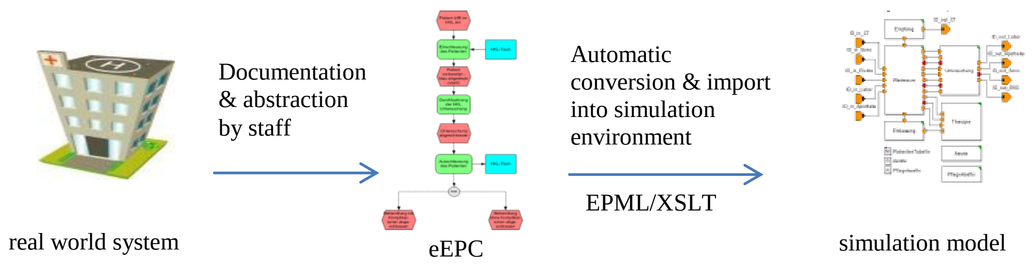
real world system
Documentation & abstraction by staff
eEPC
Automatic conversion & import into simulation environment
EPML/XSLT
simulation model

Figure 1: Schematic procedure of our approach

Within the following sections the transformation process is shown in details. The next section of this paper points out the current stage of development and outlines our approach. Following on this the initial point of transformation is described and EPML is introduced. After that, our approach is described in detail by defining transformation rules and semantic. Then the implementation of eEPC import using XSLT is described (see Figure 1).

Following on this, further requirements out of the field and restrictions of our approach are discussed and the validation of the created model and our approach is described. Upon this we make a proposal for simulation specific extensions within eEPC and EPML. The final section summarizes the results.

2. CURRENT STAGE OF DEVELOPEMENT

Within the field of model transformation there are different transformation languages known in literature, e.g. Model Driven Architecture (MDA) for software development [8]. UML plays an important role in this regard. A group of these are Graph-Transformation-Based Approaches, as UMLX [9], VIATRA - VIsual Automated model TRAnsformations [10], MOLA - Model transformation LAnguage [11] or GReAT - Graph Rewriting and Transformation language [12].

Moreover there are specific approaches for the transformation of a process model into an executable specification as it is intended in this paper, e.g. ProSiT (Process to Simulation Transformation) [13] and the continuation [14], based on hospital processes. With ProSit a concept for an intermediate transformation model is introduced, which also aims at standardization and formal rule based transformation of hospital processes. This transformation model represents an intermediate level between process and simulation model and is used to collect additional information which are needed to build up a simulation model (e.g. parameters and abstractions). The development has not completed yet. Till now the composition of the transformation model has been shown on an example of an emergency unit by using eEPC.

Ammon [15] investigated the transformation of hospital processes, described as EPC using ARIS-Toolset, to create a simulation model in the simulation system “MLDesigner”. He showed that an automatic transformation of a process structure and some additional attached knowledge into a simulation model is possible. A converter was developed that translates process descriptions in AML (xml based ARIS Markup Language) into MLDesigner simulation model described in XML. The placement of imported EPC elements is done by coordinate transformation.

Ammon found out, that the generated simulation model was not executable. The AML process descriptions did not include resources and the functional descriptions were incomplete to be transformed into an executable model. The translated model had to be modified by hand to become an executable model.

3. INITIAL POINT OF TRANSFORMATION

Initial point of process transformation is the knowledge of operational sequences and clinical processes (clinical pathways, treatment paths). This knowledge is available in documented form, e.g. as documented clinical pathway or just known by staff (e.g. doctors, nurses) without any documentation. This knowledge of the real system is necessary to build up an executable simulation model of the real system or a part of it. For building up a simulation model it is very time consuming for the model designer to collect all the needed knowledge of processes. On the other hand the knowledge owner in the real system is rarely able to put the knowledge directly into the simulation system and to build up the simulation model. Usually the necessary skills are not available to do so. Therefore, in a first step it is necessary to document the knowledge in a standardized format to work with it. It is a common and easy way to use EPC for documentation and visualization of business processes (Keller et al., 1992) and possible in the specific case of clinical pathways and hospital processes [16, pp. 55-81]. This first step of standardized documentation of the processes knowledge allows a further usage and automatic processing.

A clinical pathway defines the commonly accepted way of processing and the timeline of a patient’s treatment associated with diagnostic and therapeutic procedures. The clinical pathway, based on a diagnosis or defined cardinal symptoms, controls the treatment process and the documentation of the individual case [17, pp. 272-274].

Guided by the question how these in EPC documented clinical pathways and clinical processes can be transformed into an executable simulation model, the next section will give an overview of EPC elements and the event driven markup language (EPML) will be introduced.

4. EVENT- DRIVEN PROCESS CHAIN AND MARKUP LANGUAGE

EPC are defined as alternating series of events and functions. Events are passive elements in EPC which represent the system state. Beside that they describe the circumstances which trigger a function (e.g. patient arrived at the admission desk) and determine the variants of the function results (e.g. patient is admitted).

Functions are active elements, triggered by the above mentioned events. They model the tasks or activities within the process chain (e.g. admitting of a patient) and describe transformation from initial state (arriving of a patient) to resulting state (patient was admitted). Functions and events are connected with each other. The connection determines the control flow of the EPC. Logical connectors (rules) as OR (inclusive disjunction), XOR (exclusive disjunction) and AND (logical conjunction) are used to combine functions and events. Functions can be refined into another EPC. In this case it is called hierarchical or refined function. The basic EPC process elements are defined in the documentation by Keller et al. [18]. The extended elements of EPC (eEPC) (e.g. process link or position) are neither formalized nor well documented see Figure 2.

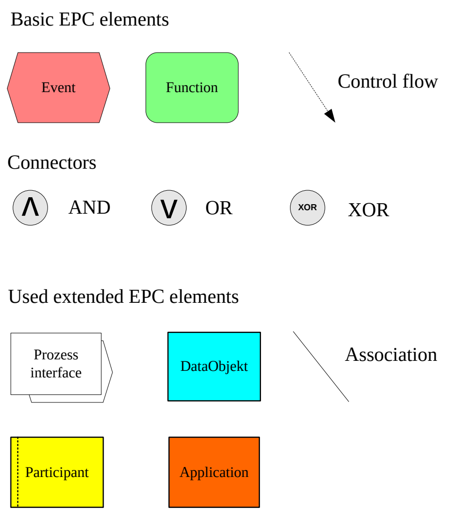

Figure 2: Overview of EPC and used eEPC process elements

Efforts of the EPC research community for standardization leads to an XML-based free and open interchange format – the EPC Markup Language (EPML) [19, 20]. It allows a standardized transfer of EPC between computer applications.

Based on this development it can be possible to transfer and import EPC automatically into a simulation system – in our case the MLDesigner (Mission Level Design Inc.). We use MLDesigner as simulation environment because we believe that for complex simulation aims a flexible and extendable environment is needed.

Corresponding concepts for converting imported xml-files into simulation blocks already exist, as shown by Ammon [15] and Helbing [22], but were not tested by simulation run based on an example out of the field and they do not use EPC-Markup Language.

A free tool with EPML support is the Oryx Editor in combination with AProMoRe (Advanced Process Model Repository). Furthermore SemTalk as Add-On for Microsoft Visio supports EPML as output format, which is used for further work. SemTalk also supports import of SAP R/3 reference model and within the EPC-Version to export content from SAP Solution Composer to Visio. The R/3 reference model contains scenario, processes, functions and components [22]. SAP software is used by almost all university hospitals in Germany in the configuration of SAP IS-H*med. This leads to further and may be better possibilities for automatic model building through export of processes out of the SAP system.

For our further work EPML in version 1.1 is used, defined in [21]. A reason therefore is, that in version 1.1 all needed features are available and the used editor only supports version 1.1. Oryx/AProMoRe editor was tested, too. Hierarchical EPC structure is not supported yet and it still seems to work with version 1.1 instead of version 2.0 (contrary to the given information).

5. PROCESS TRANSFORMATION USING eEPC

For transformation purposes the necessary (e)EPC elements need an assignment to the simulation environment. Therefore it is necessary to understand how the simulation environment works and which elements are needed.

Simulation models, as they are understood here, can be characterized by block diagrams. The functionality of a block diagram is given by the structure and linkage of the contained blocks. Within the simulation model a block is representing either a primitive or a module. A module encapsulates a further block diagram and allows the building up of a hierarchy. A primitive, as a basic block element has its own functionality, based on programming code (ptlang). Primitives and modules expose an interface consisting of input- and output ports for interconnection and communication purposes and may provide additional configuration parameters. Between connected ports the particles are moving. Particles represent events and transmit information. They are moving from output- to input ports during the simulation run. This represents the event control. This corresponds to the selected simulation domain Discrete Event (DE), which is the strongest simulation domain within the choice of domains in MLDesigner and it fits the requirements of event control mechanism.

In the following the rules for the transformation of EPC elements into components of the simulation environment are described step by step. Therefore we define simulation specific semantics for EPC elements and discuss possible implementations.

5.1 Transformation of Functions

In our approach functions will be transformed into modules in the simulation environment. There are different kinds of module types relating to their hierarchical order. A non-refined EPC function represents activity which may consume time in connection with resource usage. This basic functionality is implemented in a “function module”. A refined EPC function instantiates a module which contains another EPC. This kind of function is called “EPC module”.

5.2 Transformation of Events

The interface of a function is defined by events, which describe the working circumstances of a function and the state a function results in. Contrary to the proposal by Ammon [15], events will not be transformed into modules. Instead of this, events will become input- and output ports of function modules and determine alternatives. On behalf of this start- and end events define in- and output ports in order to connect EPC modules with each other or to connect particle sources and sinks (e.g. for graphical output in a plot). The activation of ports e.g. the arrival of a particle triggers the underlying function of a module or primitive within the simulation model.

5.3 Transformation of Logical Connectors

For branching and synchronizing within the EPC the connectors AND, OR and XOR are used. Branching connectors divide the control flow into two or more branches. This may lead to parallel processes that may require synchronization. Within an EPC there is no need for matching of branching and synchronization connectors. This causes a decision problem for the synchronization connector. A synchronizing connector is unable to know how many inputs it has to wait for [24, pp. 29 f.]. To overcome this problem, particles will be distinguished, regarding to their state between active and inactive ones. Inactive particles do not trigger a function and there is no time consumption while passing a module or primitive. They only have an effect on connectors. Following this, branching connectors within our concept will send one particle to every output port. Based on their functionality branching connectors determine which branch of the EPC will receive an active particle. All remaining branches receive an inactive particle. Out of this it is ensured that a synchronizing connector receives a particle on every input port. By this time and result of synchronization only depends on receiving particles and functionality of the synchronization connector, only using local information.

Apart from this, connectors represent control flow functionality and will be transformed into two kinds of configurable primitives – one for branching (“spread”) and one for synchronization (“sync”). During instantiation the mode of operation is specified by setting the name and parameter value according to the represented eEPC connector (AND, XOR, OR).

5.4 Transformation of Resources

Resources are essential elements of simulation models. Therefore they need to be represented within the eEPC. But there are no elements dedicated to resources within the common eEPC elements. However, there are some eEPC elements that could be interpreted as resources like participant or application (see Figure 3 - left). These elements only allow an undirected association with functions. That means they can only be allocated during the execution of a single function. It is not possible to allocate the same resource over wider model sections that consist of multiple eEPC elements. Therefore allocation- and deallocation points need to be distinguished within the eEPC, e.g. by using directed association. Looking at the available eEPC the element data object meets the requirements by allowing input and output relations (see Figure 3 - right).

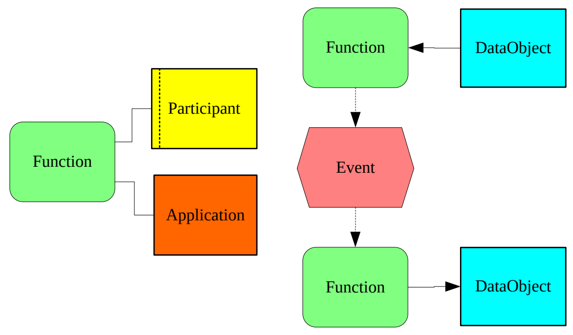

Figure 3: Modeling resources with eEPC elements

Within the simulation system (MLDesigner) resources are divided into server resources and quantity resources (for resource types see: Mission Level Design Inc, 2012). Each resource type of the MLDesigner includes its own waiting queue which is able to consider priorities. Within the simulation model the same resource can be allocated by several blocks (shared resource). Server resources are active elements and are used to delay a particle for a defined service time. When a particle enters the block, it tries to allocate the linked server resource. If the server resource is available, the particle will be delayed by defined service time. After this the resource gets deallocated and the particle leaves the block. As long as a particle uses the resource and no further processing capacity is available all other particles requesting the resource have to wait till the resource is available again. For example, server resources can be used to model medical devices as ECG device.

A quantity resource is able to provide an amount of resource units for a particle for a whole model section. Therefore resource units get allocated at a defined point within the simulation model and in case that the resource is no longer needed they will be deallocated on another point. If the capacity of the resource pool is currently depleted all other requesting particles have to wait. For example quantity resources can be used to model patient beds at a ward.

Looking at the eEPC it is possible to associate multiple resources to a function. This leads to a synchronization problem on resource allocation within the simulation system. It is not guaranteed that the linked server resources are available simultaneously, e.g. due to possible delays on resource allocation. Within the simulation system it is appropriate only to work with quantity resources. Server resources can be substituted by quantity resources in the required functionality within our approach.

Within the eEPC it is still possible to use both, participant/application and data object, to refer to the same resource. The defined transformation rule will take care, that all cases will be linked to the same quantity resource within the simulation model. As far as the associated resources are equally named within the eEPC, they are linked to the same central resource.

In case the sequence of resource allocation is important, the sequence needs to be modeled within a refined function of the eEPC. For example in case of an inpatient admission: a patient bed, a patient and a nurse are needed. To avoid unneeded resource allocation of the nurse, until the bed and the patient are allocated, resource usage must be modeled in a reasonable sequence. Normally the patient is waiting till the bed is available. Then an available nurse is bringing the patient to the bed.

In our approach each patient is represented by a particle with unique identifier within the simulation model. By this patients are passing the simulation model during simulation run. Looking at the eEPC again, it is possible to have parallel branches. Parallel branches are only meaningful to model parallel support processes that do not require the attendance of the patient. With regard to Sarshar, Dominitzki & Loos [25] an efficient modeling of elective-sequential processes is not given, when clinical pathways are modeled as EPC. This can be overcome by using parallel branches in conjunction with a dedicated patient resource. This resource is implemented as a special quantity resource that is allocated on per patient basis to ensure that a patient particle can only be part of one process step at the same time. To model elective-sequential processes a branching connector (AND/OR) is used to split involved sequential process steps into parallel branches. Each process step, that needs the patient has to receive an association to the dedicated patient resource within the eEPC. Allocation can be done as it was described for other resources. In consequence it also can be done over model sections.

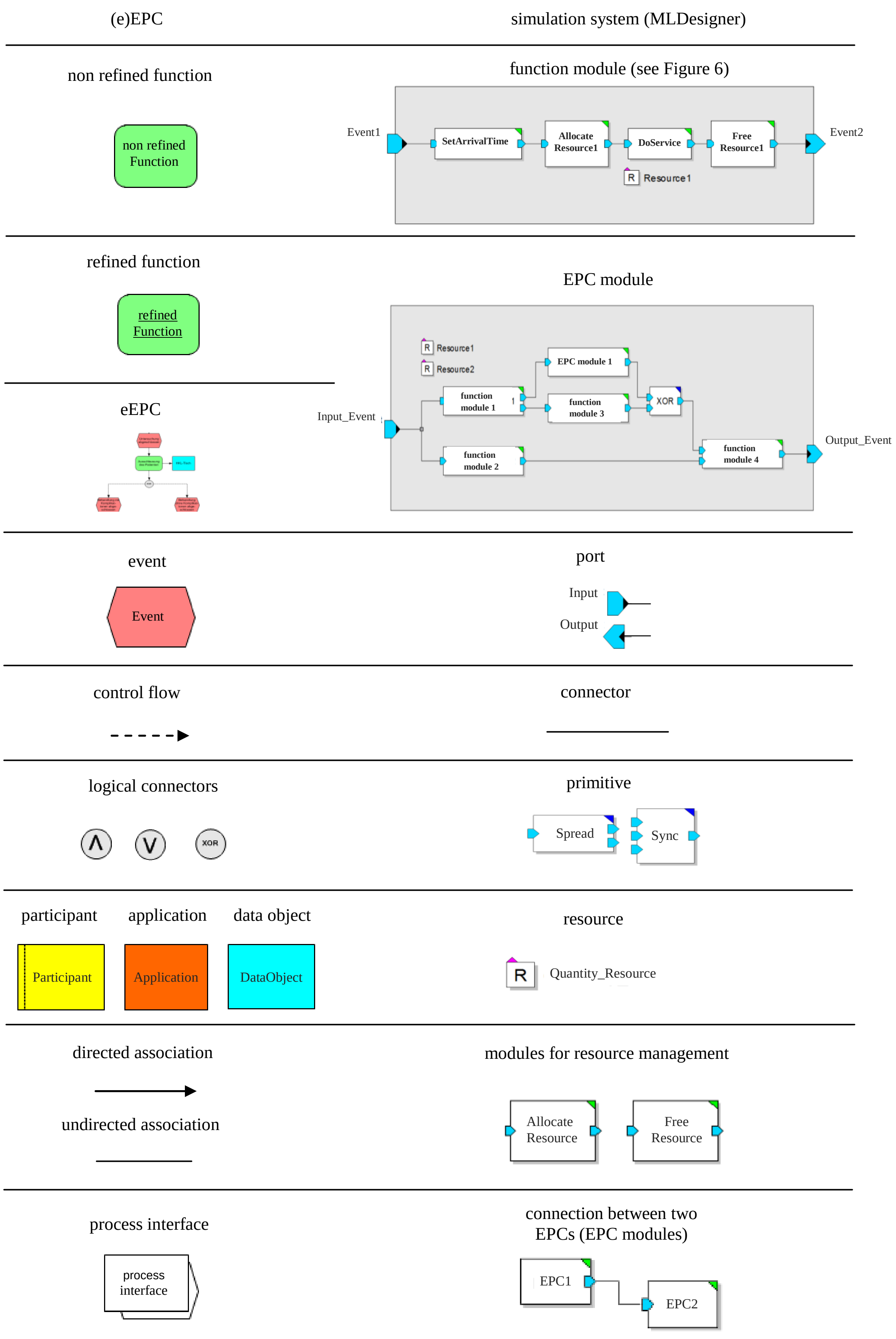

Figure 4: Mapping eEPC to MLDesigner

5.5 Transformation of Hierarchy

As mentioned above, EPCs can form a hierarchy and a simulation model is able to be set up in a hierarchy as well. There are two ways to transfer hierarchy information from the EPC to the simulation model.

- Composition (refinement):
  Upstream and Downstream events from the interface of the refined function correspond to the start and end events of the EPC on lower hierarchy level. While transforming the EPC into a simulation model, the refined function is used as an identifier.
- Concatenation:
  Concatenation means connecting multiple EPC with each other on the same level of abstraction. Therefore the element process interface is used to mark the events at the concatenation point. While transforming the EPC into a simulation model, this element is used as an identifier.

Based on this transformation rules are set up. The mapping is visualized in Figure 4.

6. IMPORT OF eEPC INTO THE SIMULATION SYSTEM BY MEANS OF EPML

For the Simulation of the processes, which are modeled as eEPC an import and conversion into executable structure of the simulation system are necessary.

For this purpose a library of basic blocks is created in the simulation system.

With relation to current requirements this library contains:

- functionality offering modules which are required in the generated eEPC functions like resource allocation or time consumption,
- primitives which are able to represent each kind of logical connectors,
- auxiliary modules and primitives to manage control flow and data structures,
- internal data structures to store the particle state and other needed information.

These basic blocks offer the needed functionality to manage the control flow and take into account the distinction in active and inactive particles within the simulation model. Then the resulting library can be utilized by a software converter to perform transformation from eEPC to the target simulation system. Instead of having to generate all needed functionality on the fly the converter is able to acquire the needed functionality by instantiating and configuring basic blocks from the library. This leads to a reduction of complexity for the transformation process.

To use these basic blocks for simulation purposes a matching internal data structure is necessary to manage the control flow. For example the internal data structure is used to store the state of particles (active/inactive) and to provide an interface for status information of the last simulated block, like arrival, completion or service time.

6.1 Generating of function modules and ports

Every non refined function of the eEPC gets transformed into its own function module with the same name. In the following step the interfaces of the function modules are determined. Therefore on the input side of the function the control flow in the eEPC will be traced back in opposite direction (upstream) until it hits upon an event. In case a connector is met, all its input flows are taken into account. The search will go on for each branch of the eEPC until an event is met. Each upstream event becomes an input port in the current function module. The same procedure is done to identify downstream events, except the control flow is followed in direction and in case that a connector is met, all output flows are considered. All found downstream events are transformed into output ports on the current function module. Each port name is derived from the corresponding event name (see Figure 5 and Figure 6).

6.2 Generating of connectors

Within the current function module all input ports get connected by using synchronizing connectors according to the connection structure of the corresponding events in the eEPC. The same happens in analog manner for the branching connectors on the output side. Connector primitives are configured and named according to their semantics (XOR, OR, AND) upon instantiation. Branching connectors use an equal weighted distribution for branching decision in initial (standard) configuration.

6.3 Generating of basic functionality and of resource usage

As it was already defined function modules may consume time or use resources as basic functionality. Therefore needed basic blocks from the library are instantiated and linked. Function modules always have a time consuming block instantiated. In opposite to this blocks for resource usage are only generated in case that at least one resource is assigned to the current function in the eEPC (see Figure 6). Each basic block that affects a resource must be linked to a shared quantity resource. Therefore quantity resources with external scope are generated and linked according to an associated data object, participant or application element. Thereby an external resource is not created as a concrete resource instance; it just provides an interface for further linkage on the next higher hierarchy level.

6.4 Generating of EPC modules and of hierarchy

A refined function references to a single eEPC. Out of this, a refined function is transformed in the same way as an eEPC. Each eEPC diagram gets transformed into an EPC module. Input- and output ports are generated according to the respective start and end events in the eEPC. Within these EPC modules the generated function modules get instantiated. The module instances may use resources that require a proper linking. Therefore external quantity resources are generated. Linking always happens in regard to the resource name. Out of this resources which are used by multiple function modules are linked to the same external resource within the EPC module.

Ports get connected regarding to their linkage within the eEPC. In an EPC module, only ports representing the same event get connected with the help of connector primitives. Connection between different events can only occur on the level of function modules.

6.5 Generating of the top level system

To simulate a model within the MLDesigner it must be instantiated within a “system model”. A system model in the MLDesigner is a top-level model and consists of instantiated primitives or modules. A system model does not have any input/output ports and cannot be instantiated. Global simulation parameters (e.g. seed for random number generation, length of simulation run) or global resources can be defined here [26].

Following this the converter generates a system model and instantiates the top level EPC module. The module instances may use resources that require a proper linking. Therefore all required resources get instantiated. Needed resources are identified by external resources of the EPC module. For each used external resource an internal resource is generated with standard parameter settings. Thereby an internal resource is a concrete resource instance; it provides resource capacity (standard capacity is one) and a waiting queue.

To use the generated model in a simulation, it is necessary to connect a particle source. Therefore a particle source in line with the current specifications and internal data structure was implemented and added to the transformation library in MLDesigner. While importing the eEPC, the source can be instantiated with a standard parameter set of particles to be emitted during simulation run (default parameterization is 1 particles every time step for 10 time steps). Connection points for sources are defined by not connected input ports of EPC modules after import of the eEPC into the simulation environment. These connection points get connected automatically with an instantiated source. To validate the simulation model, e.g. that released particles do not get stuck within a simulation run, particle sinks need to be connected. Elements from the standard library of MLDesigner are sufficient and can be connected to output ports of EPC modules that are not connected after eEPC import.

By connecting particle sources and providing default parameters the imported eEPC is now able to be executed.

6.6 Transformation on an example of an Function-Module

In the following the transformation result is shown on an example of a function module (see Figure 5 and Figure 6).

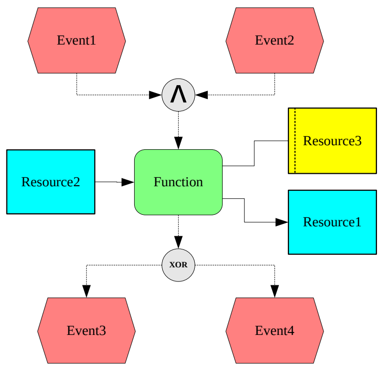

Figure 5: Example of a function with resource usage to illustrate the transformation process

Upstream events (Event1, Event2) are transformed into input ports (see left side of the function module in Figure 5) and synchronized by a Sync primitive which is configured and named as AND connector. After that the sub module “SetArrivalTime“ is used to log the arrival time of the incoming particles into the internal data structure. Then the resource allocation is done by generating an allocation module for each resource that is assigned for allocation to the respective EPC function. For each resource in use, an external resource is generated to allow a later linkage to the shared global resource. Resource allocation may consume variable amount of time for the different resources. To synchronize the resources allocation a Sync primitive, which is configured and named as AND, is used. As soon as all needed resources are allocated, a service time is generated and the particle gets delayed by module “DoService”. Also within this module status information like service time and completion time are logged into the internal data structure. Resources designated by eEPC model to be deallocated, get deallocated by generating “Free Resource” modules. The resulting particles get synchronized for reason of flow control and meet the generated connector and ports on the output side corresponding to downstream events and connector structure of the eEPC. The generated elements are placed automatically according to an in advance defined raster layout.

6.7 EPML Coding and Converting

The described transformation process is done by a software converter using XSLT (Version 2.0) style sheets based on our met transformation rules. EPML was chosen as interchange format for the eEPC diagrams. The XSLT instructions are defined in regard to the EPML notations. EPML notations are a result of an export of eEPC from the modeling Editor (SemTalk). Selected EPML notations and their equivalence within eEPC are described in the following.

An EPML file consists of collection of one or more eEPC diagrams. Each diagram is defined within an <epc> tag and has a unique identifier. Within the EPC tag all corresponding elements and edges are stored.

In the scope of EPML we find <function> and <event> tags for function and event of EPC. Control flow edges of EPC are represented by <arc> tags. Connectors also have dedicated tags <and>, <or> and <xor>.

For the elements participant, application and data object which are used as resources, EPML tags <participant>, <application> and <dataField> are provided. For association of these eEPC elements the <relation> tag is used. Also concatenation and composition are supported by EPML. The following

Figure 7 shows the example of a directed resource usage coded in EPML.

EPML in version 1.1 does not support directed associations. Therefore SemTalk is using the name attribute to distinguish between input and output relations.

The transformation is done by executing the designed XSLT-style sheets with a XSLT-processor. Therefor SAXON XSLT-processor (Saxonia Ltd.) is used. The XSLT converter is used to transfer the EPML code into the simulation system. Our approach is implemented in the MLDesigner, which is the target simulation system for transformation. To achieve the support of other target simulation systems by our converter, the transformation rules are divided into two parts. All functions related to accessing and selecting eEPC elements, like for selection of input events are stored in a separate library template. The simulation specific definitions are stored within another XSLT template. Thus it is possible to easily adapt the converter by just adding a new XSLT template with necessary definitions regarding the new simulation system.

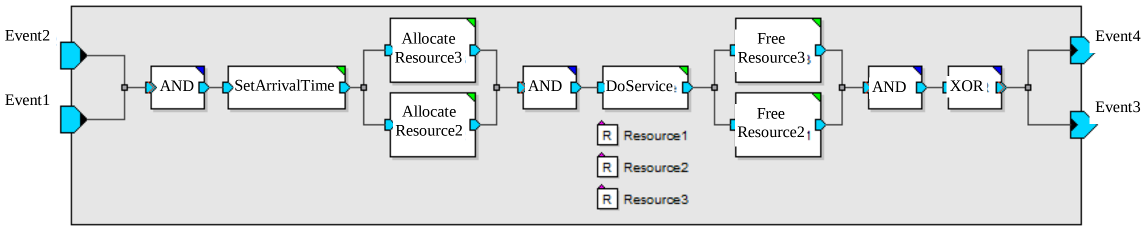

Figure 6: Resulting function module in target the simulation system

```
<dataField id="4016" defRef="769">
   <name>Entitytyp.769</name>
   <description>Resource</description>
   <graphics>
      <position height="150" width="200" x="732" y="696" />
   </graphics>
   …
</dataField>

<relation id="4015" from="4006" to="4001" name="is Input for"/>

<relation id="4017" from="4007" to="4016" name="has Output"/>
```
Figure 7: Example of a directed resource usage in EPML

7. RESTRICTIONS AND ADDITIONAL PROCESS DATA

First of all, an eEPC describes processes – in our paper these are hospital processes, especially clinical pathways. Because of that our transformation approach is limited to process models. But, that is alright for process optimization and validation purposes, as it is the aim for our further work.

Within our approach automatic model transformation is limited by the powerfulness of the used modeling language. Information, which cannot be described within the process modeling language (eEPC), cannot be transformed into a simulation model. The same applies to the used interchange format (EPML).

EPC modeling allows only limited specifications of non-refined functions. Functions are only defined by their shape and their name within the EPC. Also refining functions do not create functionality which is needed to execute a model. Even resource usage within the eEPC, which is an elementary scenario of executable process models, can only be handled by creating a specific semantic for simulation, as it is shown successfully in this paper.

Furthermore, the automatic transformation into an executable model is limited by parameters needed for the simulation run, e.g. capacity of a resource. Till now parameters within the eEPC are not specified yet, to transfer them into the simulation model. However, there are already some EPC editors existing, e.g. SemTalk, ARIS Toolset. They allow defining parameters within the EPC. Also EPML provides a possibility for user defined attributes, which can be useful, e.g. for parameters. But it should be asked whether parameters are necessary at that point. The generated simulation model may not have the final stage with regard to the simulation aims. From this point of view an implementation of parameter transfer is not done but seems to be possible for us. Within our transformation approach the generated simulation model uses default parameter sets for instantiated elements.

In many cases model transformation is the starting point for further model building of complex systems. Our approach ensures the possibility to connect a user specific central data structure to the instantiated elements of the simulation model without affecting the internal data structure. Therefore further modules are added to the MLDesigner library.

8. VALIDATION AND VERIFICATION OF THE MODEL AND THE CONCEPT

Simulation experiments should deliver reliable results to help decision making. Validation and verification are important to ensure this. Validation means whether the model is implemented correctly (also called technical validation). This also covers whether the model is operating in the intended manner - so that the eEPC is correctly transformed. The third aspect is external validation of the simulation model - whether the model correctly reflects the “real world” within the defined limitations of the model. Verification means whether the specification is met - if the implemented converter is working correctly.

There are a lot of methods for validation and verification as Page [27, pp. 145-155], Kosturiak & Gregor [28, pp. 123-125] and Müller [29, pp. 181-203] have shown. It has to be checked for each individual case which of the possibilities can be used. There is no general accepted way or method. Validation and verification can only be done by consideration of the underlying modeling purpose or question of investigation.

It is assumed, that eEPC modeling is done in line with modeling rules, in Becker, Rosemann & Schütte [30]. Modern modeling tools, as Microsoft Visio or SemTalk, support correct modeling by implemented checking rules.

In the following used methods for validation and verification for our model design and transforming approach are described. In this paper an executable model is created out of an eEPC. First of all it is the aim to transfer knowledge automatically into the simulation environment in a standardized way. At the moment there are no specific simulation aims, which should be considered in the following validation steps. To analyze specific questions the generated simulation model can be manually expanded by a user. By this the transformation result may change.

That means that there have to be two steps.

1. Validation and verification of transformation results and the correct working of the converter.
2. Validation and verification of the further usability of the generated simulation model by using it for a specific simulation purpose.

First of all the transformation result is reviewed whether the transformation rules are met, for example whether a non-refined eEPC function is transformed into a function module within the generated simulation model. After this it is reviewed whether the structure and hierarchy buildup of the generated simulation model is the same as it is in the eEPC. Within this scope the completeness of all functions, logical connectors and the usage of the same interconnections as in the eEPC are checked. As mentioned, events are transformed into ports. The next check is to find out whether all events are transformed into ports within the simulation model.

After transformation a detail check is to be done for the primitives representing logical connectors and the correct resource usage. The connectors need to be checked regarding their correct functionality and if the instantiation is in line with the original eEPC. Therefore the instantiated sources generate particles in a test run. The correct behavior of these particles is checked by passing the connectors. So it is possible to validate the right function and configuration of the connectors. This way it can also be checked, whether all particles pass the elements of the simulation model, whether the internal data structure works and whether no particles get stuck.

For validation and verification purposes a complex hospital process of a patient's stay, documented as eEPC, is used and transformed into an executable model. This process of a cardiologic department of a university hospital consists of e.g. admission of patient, several diagnosis procedures, treatment for example in a heart catheter lab, processes on cardiologic wards and discharge.

Based on these steps it was detected, that loops within the eEPC lead to a problem with particle synchronization of XOR and OR connectors at the incoming feedback branch, which was fixed manually. For example in the case that a connector has two input ports, one of it is a feedback branch, the operation mode of the connector is switched to merge. That means that incoming active particles will just pass instead of waiting for particles on all input ports. Inactive particles on the feedback branch will be destroyed. The problem is to identify feedback loops on transformation run, to perform the necessary adaption. This will need further consideration to resolve this issue automatically during the transformation process. No further faults were found.

Ongoing, the automatic generated simulation model will be used for a cardiologic department of a university hospital, to optimize the scheduling and sequencing of patients within the department, to reduce waiting time and length of stay. Therefore the created simulation model will be extended focused on this simulation aim. Based on real data (historical data), collected within the university hospital, a further (external) validation will be done. This will also show, whether the automatic generated simulation model is suitable to build up specific executable specifications as it is intended.

9. PROPOSAL FOR A SIMULATION SPECIFIC EXTENTIONS WITHIN eEPC AND EPML

The alternative usage of some elements (data object, participant, application) for modeling resources is only possible in case that these elements are not used within our modeling approach as originally intended. For example it is intended to model a data flow with the help of a data object, e.g. input and output of data during the process flow.

For further modification and development of modeling language scope of eEPC and EPML it would be desirable to have separate elements for resource usage, e.g. resource usage in only one function or the usage over wider sections with possibilities to model allocation and deallocation points.

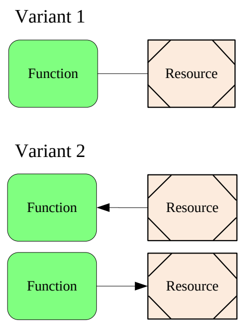

Figure 8: Use cases for a proposed resource element in eEPC (usage, allocation, deallocation)

Figure 8 shows a possible graphical realization of a proposed resource element for the mentioned two variants. The element may be connected to non-refined functions in directed or undirected fashion, this allows the allocation for just one function (variant 1) or over wider model sections that consist of multiple eEPC elements (variant 2). In EPML this proposal could lead to the introduction of a new "resource" eEPC element tag. Attributes and sub elements remain the same as for already existing tags (e.g. function, event). For interconnection possible relation types of the arc tag could be extended by "allocate", "deallocate" and "uses". This allows to model directed and undirected resource access. Resources are often shared among multiple eEPCs residing in the same EPML directory. To establish a proper link between these resource instances it is necessary to use unique identifiers throughout the model. So linked resources either need to have the same name or alternatively need to reference the same resource definition. As well the definition tag gets a new type value "resource". An example of these proposed extensions to EPML can be seen in Figure 9. The modifications of this example are consistent to EPML version 2.0.

```
<definitions>
  <definition defId="1002" type="resource">
      <name>Resource</name>
      <description>Resource
                    description</description>
  </definition>
</definitions>
…
<directory name="Root">
  <epc epcId="4" name="EPC 1">

    <function id="1">
      <name>Function</name>
      <description>Function
                    description</description>
      …
    </function>

    <resource id="2" defRef="1002">
      <name>Resource</name>
      …
    <resource>

    <arc id="3">
      <relation source="1" target="2"
             type="allocate|deallocate|uses"/>
      …
    </arc>
```

Figure 9: Integration of proposed resource element in EPML

10. SUMMARY AND OUTLOOK

On an example of hospital processes it was shown how an executable simulation model can be built up automatically out of extended event driven process chains (eEPC).

It was shown, that eEPC are suitable to model hospital processes (e.g. treatment path) and can be transformed into a model within a simulation environment. Therefore we defined the needed semantic. To make the eEPC model able to execute in a simulation run, functionality (e.g. resource usage, control flow management) is needed. We have shown how this can be added to an eEPC process model during transformation run.

In Detail a converter was developed to import eEPC models into a simulation environment (MLDesigner). As interchange format event driven process markup language (EPML) was used. Transformation rules were defined in XSLT. Based on the template structure of our style sheet design it is possible to easily adapt the converter according to the target simulation tool.

As shown the standard definitions of EPC were not sufficient to model additional requirements like resource usage. Therefore the unspecified extended EPC was used but is also not sufficient. For the moment we use the eEPC elements data object, participant and application as resource identifiers. We suggest an extension of eEPC/EPML notations for simulation purpose.

We have also shown that loops within the eEPC lead to a problem with particle synchronization of XOR and OR connectors at the incoming feedback branch. This needs further consideration on automatic transformation.

In our further work, the approach described here will be used to generate an executable model that will be extended to allow optimization with regard to scheduling and sequencing of patients within the cardiologic department of a university hospital. This will show whether our approach is also feasible to build up simulation models with specific simulation aims out of the transformation results.

Furthermore, we have a positive attitude towards the usage of standardized building blocks, which were developed in Salzwedel et al. (2007). These building blocks must be extended by our met EPC requirements and be filled up with functionality. Following this, standardized building blocks can be instantiated during the transformation process of eEPC models. This will lead to a higher level of automation, less manual adjustments and therefore faster buildup of executable specification.

11. REFERENCES

[1] Schienmann, B. 2002. Kontinuierliches Anforderungs management: Prozesse- Techniken- Werkzeuge. München, Germany: Addison-Wesley.

[2] Davis, A. M. 1993. Software Requirements: Objects, Function and States. Prentice Hall, Englewood Cliffs, N.J. USA.

[3] Salzwedel, H., Fischer, N., & Schorcht, G. 2009. Moving Design Automation of Networked Systems to Early Vehicle Level Design Stages. SAE World Congress. April 20-23. Detroit, Michigan, USA, http://www.tu-ilmenau.de/fileadmin/public/sse/Veroeffentlichungen/2009/SAE2009-H.Salzwedel.pdf.

[4] Helm, E. J., & van Oyen, M. P. 2011. Design and Optimization Methods for Elective Hospital Admissions. Paper presented at 2011 MSOM Annual Conference. June 26-28. Ann Arbor, Michigan, USA, http://sitemaker.umich.edu/jhelm/files/helm_vanoyen-elective_adm_sched.pdf.

[5] Hutzschenreuter, A. K., Bosman, P. A. N., & La Poutré, H. 2008. Enhanced hospital resource management using anticipatory policies in online dynamic multi-objective optimization. In: Pelikan, M., & Branke, J. (Ed.): Proceedings of 7th International Joint Conference on Autonomous Agents and Multiagent Systems, May 12-16, Estoril, Portugal, pp. 45-52.

[6] Kühn, M., Baumann, T., & Salzwedel, H. 2012. Genetic algorithm for process optimization in hospitals. In: Troitzsch, K. G., Möhring, M., & Lotzmann, U. (Ed.), Proceedings of 26th European Conference on Modeling and Simulation, May 29th - June 1st, Koblenz, Germany, pp. 103-107.

[7] Salzwedel, H., Richter, F., & Kühn, M. 2007. Standardized Modeling and Simulation of Hospital Processes - Optimization of Cancer Treatment Center. In: Proceedings of the International Conference on Health Sciences Simulation. January 14.-18. San Diego, California. http://www.tu-ilmenau.de/fileadmin/public/sse/Veroeffentlichungen/2007/Hospital%20Optimization%20HSS%202007.pdf.

[8] Object Management Group Inc. 2013, The Architecture of Choice for a Changing World: OMG Model Driven Architecture, http://www.omg.org/mda/ accessed 6 January 2013.

[9] Willink, E. D. 2003. UMLX : A graphical transformation language for MDA. Workshop on Model Driven Architecture Foundations and Applications, 4 September 2003. University of Twente, Netherlands.

[10] Varró, D., Varró, G., & Pataricza, A. 2002, Designing the automatic transformation of visual languages. In: Science of Computer Programming 44 (2): 205–227.

[11] Kalnins, A., Barzdins, J., & Celms, E. 2004. Model Transformation Language MOLA. In: Proceedings of MDAFA Model-Driven Architecture: Foundations and Applications. Linkoeping, Sweden, pp. 14-28.

[12] Agrawal, A., Karsai, G., & Shi, F. 2003. Graph Transformations on Domain-Specific Models. In Proceeding of OOPSLA '03 Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, October 26 – 30. Anaheim, CA, USA, pp. 386-387.

[13] Petsch, M., Schorcht, H., Nissen, V., & Himmelreich, K. 2008. Ein Transformationsmodell zur Überführung von Prozessmodellen in eine Simulationsumgebung: Modellierung betrieblicher Informationssysteme. Ges. für Informatik: Bonn, pp. 209–219.

[14] Kloos, O., & Nissen, V. 2010. Vom Prozess zur Simulation - ein Transformationsmodell-Ansatz. In: Claus, T., & Herrmann, F. (Ed.): Proceedings of Workshops „Simulation als betriebliche Entscheidungshilfe“, 14. ASIM-Fachtagung „Simulation in Produktion und Logistik“, Karlsruhe, Germany, pp. 105-119.

[15] Ammon, D. 2006. Konzipierung und Realisierung einer Software zum Import von ARIS-Modellen in MLDesigner (2006-06-01/077/IN00/2222). Germany, Ilmenau: Technical University of Ilmenau.

[16] Eiff von, W. 2001. Geschäftsprozeßmanagement: Methoden und Techniken für das Management von Leistungsprozessen im Krankenhaus. Gütersloh, Germany: Verl. Bertelsmann-Stiftung.

[17] Wenzlaff, P. 2006. Glossar zum Prozessmanagement im Gesundheitswesen und zu Integrierten Behandlungspfaden: Praxishandbuch Integrierte Behandlungspfade. Heidelberg, Germany: Economica.

[18] Keller, G., Nüttgens, M., & Scheer, A.-W. 1992. Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". Inst. für Wirtschaftsinformatik, Univ. Saarbrücken, Germany.

[19] Mendling, J. & Nüttgens, M. 2003. XML-basierte Geschäftsprozessmodellierung. In: Uhr, W., Schoop, E., & Esswein, W. (Ed.), Proceedings der 6. Internationalen Tagung Wirtschaftsinformatik: Medien - Märkte – Mobilität, Heidelberg, Germany: Physica, pp. 161-180.

[20] Mendling, J., & Nüttgens, M. 2005. EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-DrivenProcess Chains (EPC). Technical Report. Wien, Austria.

[21] Mendling, J., & Nüttgens, M. 2004. Exchanging EPC Business Process Models with EPML, In: Nüttgens, M., & Mendling, J. (Ed.), Proceedings of the 1st GI Workshop XML4BPM - XML Interchange Formats for Business Process Management, Marburg, Germany, pp. 61-79.

[22] Helbing, D. 2008. Konzeption und Realisierung eines Konverters zur Übertragung von MoBimeP-Strukturen als Referenzen für MLDesigner-Modelle. Germany, Ilmenau: Technical University of Ilmenau.

[23] SemTalk GmbH 2012. Tutorial SemTalk Version 4.0: EPK Edition. http://www.semtalk.com/pub/tutsem40epcg.pdf, accessed 4 April 2012

[24] Rittgen, P. 2000. Quo vadis EPK in ARIS? - Ansätze zu syntaktischen Erweiterungen und einer formalen Semantik. In: König, W., Mertens, P., Hasenkamp, & U. et al. (Ed.), Wirtschaftsinformatik 42 (1): 27-35.

[25] Sarshar, K., Dominitzki, P., & Loos P 2005. Einsatz von Ereignisgesteuerten Prozessketten zur Modellierung von Prozessen in der Krankenhausdomäne. In: Nüttgens, M., & Rump, F. J. (Ed.), Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. In:Proceedings 4. GI-Workshop und –Arbeitskreistreffen 167, pp. 97-116.

[26] Mission Level Design Inc. 2012. User’s Manual ML Designer version 2.8., http://www.mldesigner.com/fileadmin/assets/MLDesigner/Manual/manual.pdf, accessed 18 December 2012.

[27] Page, B. 1991. Diskrete Simulation: Eine Einführung mit Modula-2. Berlin, Germany.

[28] Kosturiak, J., & Gregor, M. 1995. Simulation von Produktionssystemen. Wien, Austria: Springer.

[29] Müller, J.-A. 1998. Simulation ökonomischer Prozesse. Wien, Austria.

[30] Becker, J., Rosemann, M., & Schütte, R. (1995). Grundsätze ordnungsmäßiger Modellierung. In Wirtschaftsinformatik 37 (5): 435-445