UDC 004.75

MAPPING BUSINESS PROCESSES MODELS FROM PETRI NETS INTO EVENT-DRIVEN PROCESS CHAINS

S. Zlatkin, R. Kaschek

Department of Information Systems,
Massey University, New Zealand
S.Zlatkin|R.H.Kaschek@massey.ac.nz

Business Process Reuse is an important phase in the Business Process life-cycle that for some reason has not attracted enough attention. To support the reuse phase we are building a software infrastructure called the Process Assembler. Process reuse involves many steps. In the paper we briefly discuss all of them, but then focus only on the mapping from one process definition language (which is Petri Nets) into another (Event-driven Process Chains). We build the mapping algorithm and support it by software, which is the core of the Mapper component of a system that is currently under construction. An example demonstrating the work of the algorithm is also provided.

Introduction

Organizations require material resources, human resources, and business processes to produce goods or services. According to [WM99] business process is "*a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships*". A market exists for material and human resources. Currently no market exists for business processes. Aiming at creating such a market leads to the problem of process reuse. Process reuse requires the existence of an environment in which the process quality can be assessed, business deals negotiated and prices determined. The Process Assembler, the system we are building, is supposed to be a software component that provides key functionality for business process reuse.

Our initial heuristic for business process reuse is: (1) build-up domain specific process libraries, (2) limit modifications of the process structure to a sensible degree, (3) take into account modifying the resources involved in a business process, (4) use an effective and efficient business process specification and access facility, (5) implement a reuse infrastructure that is based on a reuse organization model such as the two-library model for software reuse (see, for example [Gr98]), and (6) put in place workable models of process pricing and usage.

The key functionalities of the Process Assembler are:
– **Adapt** a business process to the case at hand.
– **Assess** a business process as to whether or not it should be reused in the case at hand.
– **Map** a business process, i.e., translate it from a language L into a language L'.
– **Display** to a user the available business processes in a suitable form that can be parameterized by the user.
– **Navigate** the displayed business processes.
– **Retrieve** the business processes that best match a given specification.
– **Specify** a set of business processes. We assume currently that respective specifications will be obtained in terms of an SQL-like dedicated query language.

These key functionalities of business process reuse cover the stages of workflow type reuse as given in [Kr00].

In this paper we mainly focus on some aspects of mapping function of the Process Assembler.

The rest of the paper is structured as follows: in the next section we briefly discuss the Process Assembler and show its architecture. In section three we focus on some aspects of business process mapping. We describe Petri Nets in subsection 3.1 and Event-driven Process Chains in subsection 3.2 as they are among the best developed and accepted process modeling techniques. We derive the mapping algorithm in subsection 3.3. This algorithm is a part of the Process Assembler's Mapper component. In section 4 we provide an example illustrating how the algorithm works. Finally, we conclude the paper and discuss our future work in section 5 and provide our references in section 6.

1 Process Assembler

The Process Assembler (PA) itself is a Web-based information system for business process reuse. It's most basic functionality is thus the recording, storing, retrieving, and disseminating of processes, see, e.g. [Hi95]. The PA distinguishing qualities are (1) it is a model based component that allows the registration of process modeling languages that comply to our process modeling language meta model; (2) it implements associative (i.e. semantics based) retrieval of stored processes; (3) it enables users after the provisioning of the respective translation procedures to translate a process from its encoding in a process modeling language L into an encoding in process modeling language L'. For

enabling associative retrieval of processes the PA firstly has a built-in process meta-model that involves process goals which is derived from [Ka95]; and secondly integrates a thesaurus the lexical relations (for that term see, e.g. [Yu96]) of which can be exploited in the process query language under development. In Fig. 1 we recall the high-level PA-architecture from [ZK05]. This architecture complies with the repository system architecture requirements of [Ja04].

The Registry is for registering business process modeling languages with the PA. Obviously the Registry is a key component of the PA's model based architecture and it needs to provide to the other components the functionality required for handling the processes. It needs to provide the Repository the meta-information required to store and retrieve business processes. Obviously also the Modeler relies on that information, as it needs to be capable of processing processes. The Assessor component aids users in qualitatively assessing the reusability of a business process

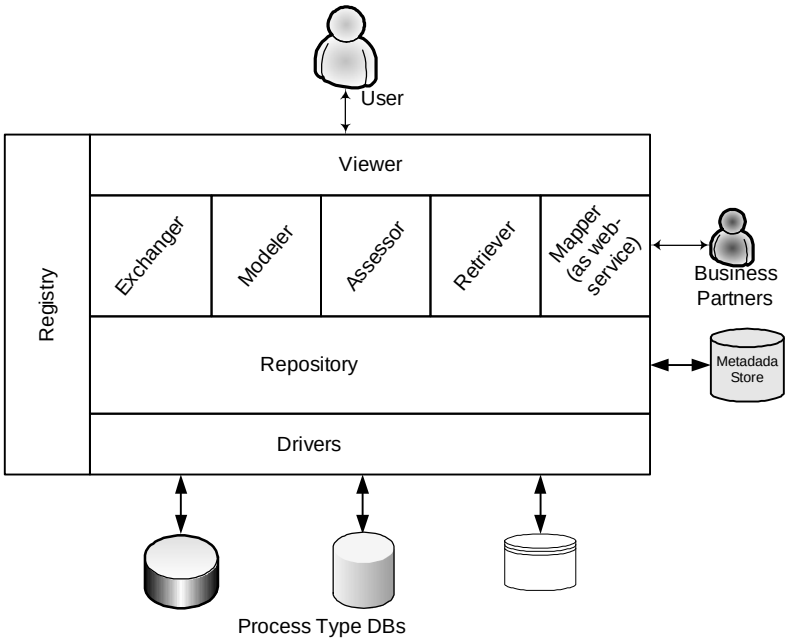

Fig.1. The Process Assembler Architecture

in the case at hand. Here we intend using ripple down rules for representing and acquiring the assessment knowledge. These are "if-then-else rules" that can be refined by exceptions and can be cascaded via the exception branch or the else branch, see, for example [PH06]. It occurs such that due to their simplicity and their limited composition principle for representing knowledge using ripple-down-rules does not imply that a knowledge engineer will be needed. Finally, the Exchanger, the Mapper, and the Viewer are for importing and exporting business processes, translating business processes, and for having an overview of the processes stored in the repository. Below we deal with the Mapper component and translating processes from one process definition language to another.

There exist a number of available process definition languages [MO04]. Some of them are vendor specific (e.g. WDL [Gr03], etc.), some are de-facto standards [Ha05] created and promoted by different standardization organizations (e.g. UML Activity Diagrams by OMG [OMG03], XPDL by Workflow Management Coalition [XPDL,WM99], etc.), some are well accepted for modeling processes for a particular class of problems (e.g. Petri Nets [AH02], Event-driven Process Chains [Ke92], etc.). Below in this paper we focus on mapping from Petri Nets into Event-driven Process Chains.

In the next section we first introduce those two languages, discuss their fields of applications, pros and contras. And then we come up with the actual mapping algorithm.

2. Mapping from Petri Nets into Event-driven Process Chains

2.1 Modeling Business Processes with Petri Nets. The theory of Petri Nets was developed in 60s. It is based on classical graph theory and is an extension for theory of final state machines. Since then Petri Nets are used for modeling complex system behavior, from protocols and hardware systems to economic, physical and social systems including business process modeling. Petri Net is built on 2 notions: events (actions taking place in the system) and conditions (logical description of system state). Exactly, system state controls occurrence of events. In Petri Nets conditions are modeled by Places, and Events are modeled by

Transition input is a precondition of a given event, while output is a post-condition respectively. Places and transitions are connected by directed arcs. Occurrence of event means start of respective transition. Satisfying event is

represented by marker in the respective place. Transition start removes all markers, which compose a precondition, and creates new markers, which compose a post-condition.

Formally Petri Net can be defined as a three-tuple $PN = \langle P, T, R \rangle$, where P is a set of places, T is a set of transitions, and R is binary flow relation $R \subseteq (P \times T) \cup (T \times P)$.

According to [MS] main advantages of Petri Nets are:

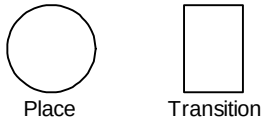

Fig. 2. Petri Nets Modeling concepts

- **Formal Semantics**. Business process defined in terms of Petri Nets has clear precise view as semantics of classic Petri Nets is defined formally.
- **Graphical Notation**. The instrument of Petri Nets represents system of graphic designations by means of which it is possible to make respective graphs.
- **Properties**. The basic properties of Petri Nets are studied during many years. The developed mathematical methodology exists. It allows analyzing and investigating these properties when respective net is constructed.
- **Analysis**. Petri Nets differ by existing of several analysis methods.

The following can be considered as other features of Petri Nets:

- **Simultaneity**. In Petri Nets model two resolved non-interacting events can occur independently from each other. There is no necessity to synchronize event while it is not required by modeled system. But, when synchronization is necessary, to model it is easy.
- **Asynchronous nature**. In Petri Nets the concept of time is absent. It means that there is no rigid time order of events that reflects a real life. The structure of Petri Nets comprises all necessary information for definition of possible sequences of events.
- **Non-determination**. The order of occurrence of events is one of possible, supposed by the basic structure. If during any moment of time more than one transition is authorized, any of several possible transitions can become "the next" that started. The choice of started transition is carried out by non-determined way, i.e. randomly. This feature of Petri Nets reflects that fact, that in a real vital situation, where some actions occur simultaneously, the arising order of occurrence of events is not unequivocal; more likely any of possible event sequences can occur. However, the partial order of occurrence of event is unique.
- **Primitive events**. Start of transition (event) is considered as the instant. That is, the probability of occurrence of two and more events simultaneously is equal to zero. Non-primitive events are broken to *n* primitive, or represent them in the form of squares (when necessary for understanding of system as a whole), and primitive - in the form of laths.

Thus, Petri Nets are ideal means for modeling of those processes or systems in which events occur asynchronously and independently, and systems with the distributed control in which some business processes are carried out simultaneously.

As shown in [MO04] using Petri Nets in an explicit form for business process modeling is not suitable because graphical notation is not intuitively clear and because it is not possible to model all types of processes. Though, a number of modeling languages, such as XPDL [XPDL], developed by Workflow Management Coalition, and Petri Nets Markup Language, used for interchanging of Petri Nets Markup Language, are developed based on Petri Nets. Also on the base of Petri Nets new process and workflows modeling language was created - YAWL [AH03] (Yet Another Workflow Language).

2.2 Modeling Business Processes with Event-driven Process Chains. The strength of Event-driven Process Chains (EPC) lies on its easy-to-understand notation that is capable of portraying business information system while at the same time incorporating other important features such as functions, data, organizational structure and information resources as already described before. This makes EPC a widely acceptable standard to denote business processes. In the following the elements used in EPC diagram will be described:

- **Event**. Events are passive elements in EPC. They describe under what circumstances a function or a process works or which state a function or a process results. Examples of events are “requirement captured”, “material on stock”, etc. In the EPC graph an event is represented as hexagon.
- **Function**. Functions are active elements in EPC. They model the tasks or activities within the company. Functions describe transformations from an initial state to a resulting state. In case different resulting states can occur, the selection of the respective resulting state can be modeled explicitly as a decision function using

logical connectors. Functions can be refined into another EPC. In this case it is called hierarchical function. Examples of functions are capture requirement, check material on stock, etc. In the EPC graph a function is represented as rounded rectangle.

- **Organization unit**. Organization units determine which person or organization within the structure of an enterprise is responsible for a specific function. Examples are sales department, sales manager, procurement manager, etc. It is represented as an ellipse with a vertical line.
- **Information, material, or resource object**. In the EPC the information, material, or resource objects portray objects in the real world, for example business objects, entities, etc., which can be input data serving as the basis for a function, or output data produced by a function. Examples are material, order, etc. In the EPC graph such an object is represented as rectangle.
- **Process path**. Process paths serve as navigation aid in the EPC. They show the connection from or to other processes. A process path is represented in EPC as a function and event symbol (function symbol in front of event symbol).
- **Control flow**. A control flow connects events with functions, process paths, or logical connectors creating chronological sequence and logical interdependencies between them. A control flow is represented as a dashed arrow.
- **Logical connector**. In the EPC the logical relationships between elements in the control flow, that is, events and functions, are described by logical connectors. With the help of logical connectors it is possible to split the control flow from one flow to two or more flows and to synchronize the control flow from two or more flows to one flow. There are three kinds of logical relationships defined in EPC:
  - *Branch/Merge*. Branch and merge correspond to making decision of which path to choose among several control flows. A branch may have one incoming control flow and two or more outgoing control flows. When the condition is fulfilled, a branch activates exactly only one of the outgoing control flows and deactivates the others. The counterpart of a branch is a merge. A merge may have two or more incoming control flows and one outgoing control flow. A merge synchronizes an activated and the deactivated alternatives. The control will then be passed to the next element after the merge. A branch in the EPC is represented by an opening XOR, whereas a merge is represented as a closing XOR connectors.
  - *Fork/Join*. Fork and join correspond to activating all paths in the control flow concurrently. A fork may have one incoming control flow and two or more outgoing control flows. When the condition is fulfilled, a fork activates all of the outgoing control flows in parallel. A join may have two or more incoming control flows and one outgoing control flow. A join synchronizes all activated incoming control flows. In the EPC diagram how the concurrency achieved is not a matter. In reality the concurrency can be achieved by true parallelism or by virtual concurrency achieved by interleaving. A fork in the EPC is represented by an opening AND, whereas a join is represented as a closing AND connectors.
  - *OR*. An OR relationship corresponds to activating one or more paths among control flows. An opening OR connector may have one incoming control flow and two or more outgoing control flows. When the condition is fulfilled, an opening OR connector activates one or more control flows and deactivates the rest of them. The counterpart of this is the closing OR connector. When at least one of the incoming control flows is activated, the closing OR connector will pass the control to the next element after it.
- **Information flow**. Information flows show the connection between functions and input or output data, upon which the function reads, changes or writes.
- **Organization unit assignment**. Organization unit assignments show the connection between an organization unit and the function it is responsible for.

Formally event-driven process chain can be defined as a four-tuple $EPC = \langle E, F, C, D \rangle$ (here we consider only the control aspect of the EPC notation), where E is a set of events, F is a set of functions, C is a set of logical connectors, and D is flow relation (a set of arcs) such that $D = (E \times F) \cup (F \times E) \cup (E \times C) \cup (C \times E) \cup (F \times C) \cup (C \times F) \cup (C \times C)$.

We additionally distinguish three subsets $X \subset C, A \subset C, O \subset C$ of XOR-, and-, and or- connectors respectively.

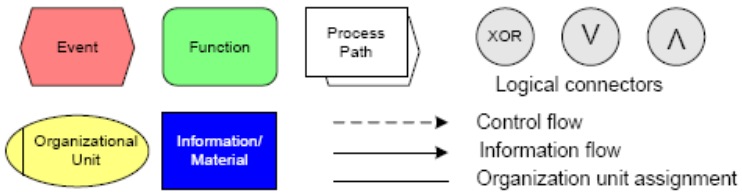

Fig. 3. EPC Modeling Concepts

After knowing the elements in the EPC, the next step is how to use these elements to model a business process using EPC. The following are some hints and constraints in connecting these elements to form an EPC diagram (a formal description of syntactical correctness of an EPC diagram will be described later):

- An event can only be followed by a function.
- A function has always a following event.
- In a single EPC graph without process paths, the graph must have at least one start event and one end event. In other words, a graph must be started and ended with events, not with functions.
- If the graphs have process paths that link them, in the source graph the process path should be put at the end of the graph after the end event, and in the target graph the process path should be put at the beginning of the graph before the start event.
- A combination of functions and events can be achieved using logical connectors. Logical connectors can be placed between functions on the one hand and events on the other hand, but the alternation of functions and events must always be maintained.
- An event is a passive element
- A function is an active element
- Logical connectors should match, meaning that an opening XOR serving as a branch should be closed by another XOR connector. The same rule applies to fork/join using AND connector and OR connector.
- All elements must be connected to the control flow, because an isolated element would have no meaning or contribution to the whole process.

In order to be able to exchange EPC between different tools and systems Mendling and Nüttgens (see [MN04]) have developed an XML-based interchange format for Event-driven Process Chains called the EPC Markup Language (EPML).

3.3 Mapping Algorithm. To achieve mapping we follow some recommendations of Aalst given in [Aa98] but use them in opposite directions (i.e. from Petri Nets to EPC). In this it is suggested to map places to events as both are passive elements, and transitions to functions as active ones (see Fig. 4).

Mapping the binary flow relation of Petri Nets contains six cases. Sequences of places and transitions (see Fig. 5-a and 5-b) are mapped one-to-one as sequences of events and functions respectively.

In case a place is followed by several transitions (see Fig.5-c) only one may fire. That means the choice should be made. The choice in EPC is modeled by the use of XOR-splitter. When a place is a result of several transitions (see Fig. 5-e), it is taken when any (the first, to be more precise) of the transitions fires. In EPC it is modeled by the use of XOR-joiner.

In case a transition is followed by several places (see Fig. 5-d) all the places are taken. That is the case of parallelism which is in EPC modeled by the use of AND-splitter. Transition following several places (see Fig. 5-f) fires when all the places are taken. In EPC it is modeled by AND-joiner.

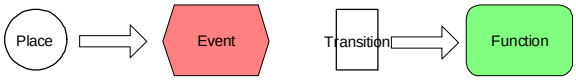

Fig. 4. Mapping basic element

To start with the mapping algorithm we first make an assumption that a Petri Net starts and terminates with a place or places, i.e. $\forall t \in PN,\ {}^\bullet t \neq \varnothing,\ t^\bullet \neq \varnothing$, where ${}^\bullet t$ is the pre-domain of *t* (i.e. the set of places followed by a transition *t*), and $t^\bullet$ is the post-domain of *t* (i.e. a set of places following a transition *t*). We also define that initially sets of events, functions and logical connectors for the target EPC are empty as well as sets of places and transitions that have been already mapped during the algorithm execution (step 1).

We define P_0 as a set of initial places (step 1.1). For every place in that set we introduce a respective event (according to recommendation given above) and add it to a set of events of the target EPC (step 1.2). We use the symbol $\Rightarrow$ to demonstrate the introduction routine.

If P_0 is empty, which means that the net is empty, or $p^\bullet$ is empty, which means that we reach the final place, algorithms stops (step 2). Otherwise we define the post-domain of all initial places (step 2.1) and for each transition in this set that has not been yet mapped (because of possible loops) we introduce a respective function and add it to a set of functions of the target EPC (step 2.2). We then check how many places are followed by each not yet mapped transition from a set defined on step 2.1, and how many transitions follow each place from P_0. In case those numbers greater than 1 we introduce AND-joiner and "connect" it with the respective function (step 2.3) or we introduce XOR-splitter and "connect" the respective event with it (step 2.4) respectively. Then we define events to be joined (step 2.5) and

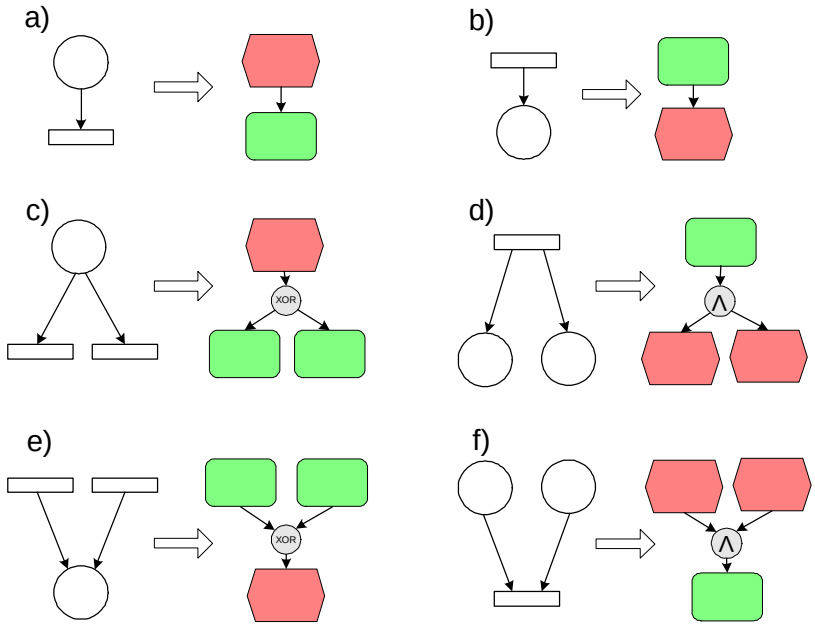

Fig. 5 – Mapping control flows

functions to be chosen (step 2.6). In step 2.7 we add places that have been just mapped to a special set P' and define a set of not-mapped transitions that should be mapped next.

Step 3 (including all sub-steps) is analogical to step 2 (we rather swap places and transitions and AND- and XOR- connectors).

After step 3 is complete we return to step 2, check whether we reach the final places and continue if we not.

The full algorithm is the following:

1. $P' = \varnothing, T' = \varnothing, E = \varnothing, F = \varnothing, C = \varnothing$

1.1 $P_0 = \{ p : {}^\bullet p = \varnothing \}$

1.2 $\forall p \in P_0 \Rightarrow e_p \in E$

2. If $(P_0 = \varnothing)$ or $(p^\bullet = \varnothing)$ then *exit* else:

2.1 $T_1 = \{ t : t \in p^\bullet, \forall p \in P_0 \}$

2.2 $\forall t \in T_1 \setminus T' \Rightarrow f_t \in F$

2.3 $\forall t \in T_1 \setminus T', |{}^\bullet t| > 1 \Rightarrow a_t \in A ; (a_t, f_t) \in D$

2.4 $\forall p \in P_0, |p^\bullet| > 1 \Rightarrow x_p \in X ; (e_p, x_p) \in D$

2.5 $\forall e_p, (e_p, x_p) \notin D, (p, t) \in R$

$$\text{If } |{}^\bullet t| = \begin{cases} 1 \Rightarrow (e_p, f_t) \in D \\ n, n > 1 \Rightarrow (e_p, a_t) \in D \end{cases}$$

2.6 $\forall e_p, (e_p, x_p) \in D, (p, t) \in R$

$$\text{If } |{}^\bullet t| = \begin{cases} 1 \Rightarrow (x_p, f_t) \in D \\ n, n > 1 \Rightarrow (x_p, a_t) \in D \end{cases}$$

2.7 $P' = P_0 \cup P'$

2.8 $T_1 = T_1 \setminus T'$

3.1 $P_1 = \{ p : p \in t^\bullet, \forall t \in T_1 \}$

3.2 $\forall p \in P_1 \setminus P' \Rightarrow e_p \in E$

3.3 $\forall p \in P_1 \setminus P', |{}^\bullet p| > 1 \Rightarrow x_p \in X ; (x_p, e_p) \in D$

$$3.4\ \forall t \in T_1, |t^{\bullet}| > 1 \Rightarrow a_t \in A\ ; (f_t, a_t) \in D$$

$$3.5\ \forall f_t, (f_t, a_t) \notin D\ , (t, p) \in R$$

$$\text{If } |{}^{\bullet}p| = \begin{cases} 1 \Rightarrow (f_t, e_p) \in D \\ n, n > 1 \Rightarrow (f_t, x_p) \in D \end{cases}$$

$$3.6\ \forall f_t, (f_t, a_t) \in D\ , (t, p) \in R$$

$$\text{If } |{}^{\bullet}p| = \begin{cases} 1 \Rightarrow (a_t, e_p) \in D \\ n, n > 1 \Rightarrow (a_t, x_p) \in D \end{cases}$$

$$3.7\ T' = T_1 \cup T'$$

$$3.8\ P_0 = P_0 \setminus P'$$

Loop to 2.

3. Example

To demonstrate how the algorithm works lets have a look at the simplified process of annual leave application. After a respective process was initiated employee creates application and submits it to the Head of Department (HoD). HoD in turn approves it or rejects (we do not consider the situation when the application may be returned to the employee for editing). If the application is rejected employee is notified about the decision. In case it is approved HoD first proceeds the application (issues an order and notifies human resource section) and then notifies the employee. Petri Net in Fig. 6 shows that process, where p_0 means "process started", p_1 – "application created", p_2 – "application submitted", p_3 – "application checked", p_4 – "order issued", p_5 – "human resources notified", p_6 – "application finished", t_1 means "create application", t_2 – "submit application", t_3 – "check application", t_4 – "proceed", t_5 – "notify employee", t_6 – "notify employee", *OK* means application is approved, *KO* means application is rejected.

$PN = \langle P, T, R \rangle$

$P = \{p_0, p_1, p_2, p_3, p_4, p_5, p_6\}$

$T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$

$R = \{(p_0, t_1), (t_1, p_1), (p_1, t_2), (t_2, p_2), (p_2, t_3), (t_3, p_3), (p_3, t_4), (p_3, t_6), (t_4, p_4), (t_4, p_5), (p_4, t_5), (p_5, t_5), (t_5, p_6), (t_6, p_6)\}$

By applying the algorithm from the previous section we get the following EPC:

$EPC = \langle E, F, C, D \rangle$

$E = \{e_{p0}, e_{p1}, e_{p2}, e_{p3}, e_{p4}, e_{p5}, e_{p6}\}$

$F = \{f_{t1}, f_{t2}, f_{t3}, f_{t4}, f_{t5}, f_{t6}\}$

$C = \{x_1, a_1, a_2, x_2\}$

$D = \{(e_{p0}, f_{t1}), (f_{t1}, e_{p1}), (e_{p1}, f_{t2}), (f_{t2}, e_{p2}), (e_{p2}, f_{t3}), (f_{t3}, e_{p3}), (e_{p3}, x_1), (x_1, f_{t4}), (x_1, f_{t6}), (f_{t4}, a_1),$
$(a_1, e_{p4}), (a_1, e_{p5}), (e_{p4}, a_2), (e_{p5}, a_2), (a_2, f_{t5}), (f_{t5}, x_2), (f_{t6}, x_2), (x_2, e_{p6})\}$

The graphical view of this chain is given in Fig. 7. It is now possible to edit the EPC for further automation. That includes assigning organization units responsible for each function, specify documents, etc.

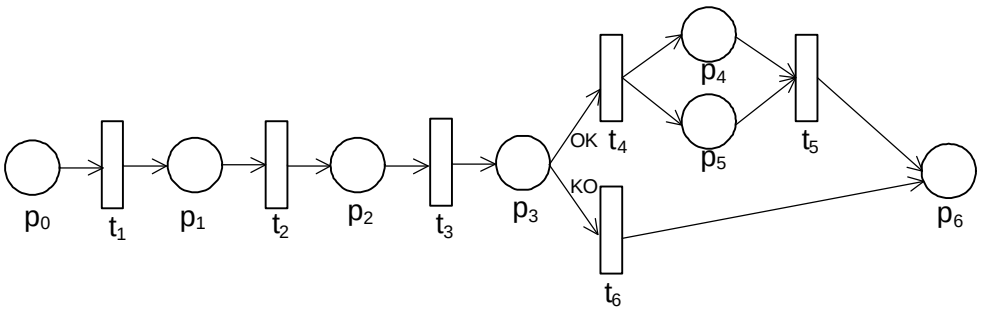

Fig. 6 – Petri Net for Annual Leave Application Process

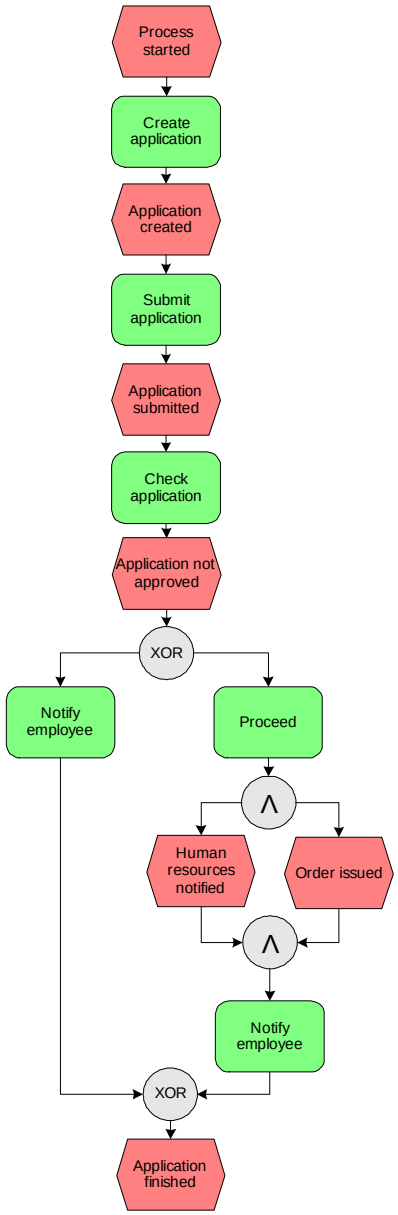

Fig. 7 – Event-driven Process Chain for
Annual Leave Application Process

Conclusions and Future work

In this paper we have derived an algorithm for mapping process models presented as Petri Nets to Event-driven Process Chains. We support this algorithm by software developed. That software is a part of the Mapper component of the Process Assembler, the system aimed to support Business Process Reuse.

We are currently finishing algorithm and its software realization for mapping process models given in EPC to WDL, the internal language of "@enterprise" [Gr03] workflow management systems. Once created we will have support for mapping on all phases of business process life-cycle, i.e. from conceptual modeling through simulation and detailed modeling to automation within workflow management system.

1. [Aa98] van der Aalst, W.M.P.: Formalization and Verification of Event-driven Process Chains. Computing Science Reports 98/01, Eindhoven University of Technology, Eindhoven, 1998.
2. [AH02] van der Aalst, W.M.P.; van Hee, K.: Workflow management: Models, Methods, and Systems, MIT Press, 2002.
3. [AH03] van der Aalst, W.; ter Hofstede, A.: YAWL:Yet Another Workflow Language, QUT Technical report, FIT-TR-2003-04, Queensland University of Technology, Brisbane, 2003.
4. [Gr98] Graham, I.: Requirements Engineering and Rapid Development: A Rigorous Object-Oriented Approach, Addison-Wesley, 1998.
5. [Gr03] Groiss Informatics: System Administration, Dokumentversion 6.1.1, URL: http://www.groiss.com, 2003.
6. [Ha05] Havey, M.: Essential Business Process Modeling, O'Reilly, ISBN: 0-596-00843-0, August 2005.
7. [Hi95] Hirschheim, R.; Klein, H. K.; Lyytinen, K.: Information Systems Development and Data Modeling, Conceptual and Philosophical Foundations, Cambridge University Press, 1995.
8. [Ja04] Jablonski, S.; Petrov, I.; Meiler, C.; Mayer, U.: Guide to Web Application and Platform Architectures, Springer, 2004.

9. [Ka95] Kaschek, R.; Kohl, C.; Mayr, H.: Cooperations - An Abstraction Concepts suitable for Business Process Reengineering, In: ReTIS'95, OCG, 1995.
10. [Ke92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeβmodellierung auf der Grundlage "Ereignis- gesteuerter Prozeβketten (EPK)", in: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken 1992.
11. [Kr00] Kradolfer, M.: A Workflow Metamodel Supporting Dynamic, Reuse based Model Evolution, PhD Thesis, University of Zurich, 2000. URL:
12. http://www.ifi.unizh.ch/ifiadmin/staff/rofrei/Dissertationen/Jahr 2000/thesis kradolfer.pdf
13. [MS] Martynova, O.; Shundeev, A.: Modeling and Analysis of Business Processes using Petri Nets (in Russian).
14. [MN04] Mendling, J.; Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: M. Nüttgens, J. Mendling, eds.: Proc. of the 1st GI Workshop XML4BPM - XML Interchange Formats for Business Process Management" at Modellierung 2004, Marburg Germany, pages 61-79, March 2004.
15. [MO04] Miheev, A.; Orlov, M.: Perspective of Workflow Management Systems, PC Week, 2004. URL: http://www.pcweek.ru (in Russian).
16. [OMG03] Object Management Group, OMG Unified Modeling Language Specification, Version 1.5, http://www.omg.org/technology/documents/formal/uml.htm , March 2003.
17. [PH06] Pham, S. B.; Hoffmann, A.: Intelligent Support for Building Knowledge Bases for Natural Language Processing, In: Intelligent Assistant Systems, Idea Group, 2006.
18. [WM99] Workflow Management Coalition Terminology and Glossary, Document Number WFMC-TC- 1011, February 1999.
19. [XPDL] Workflow Process Definition Interface - XML Process Definition Language, Document Number WFMC-TC-1025, URL: http://ww.wfmc.org.
20. [Yu96] Yule, G.: The Study of Language, Cambridge University Press, 1996.
21. [ZK05] Zlatkin, S.; Kaschek, R.: Towards Amplifying Business Process Reuse, In: ER 2005 Workshops, LNCS volume 3770, Springer, 2005.