INTEGRATING BUSINESS PROCESSES AND INFORMATION SYSTEMS

Giorgio Bruno
Dip. Automatica e Informatica, Politecnico di Torino, Torino, Italy

Keywords: Business processes, enterprise information systems, human tasks, control flow, information flow.

Abstract: While the need for a better integration between business processes and enterprise information systems is widely acknowledged, current notations for business processes are inclined to emphasize control-flow issues and omit to provide adequate links with two fundamental aspects of enterprise information systems, i.e. the human tasks and the information flow among the tasks. This paper presents a notation for business processes whose purpose is to overcome the above-mentioned limitations. This notation, called tk-nets (task-oriented nets) supports four interaction patterns between process elements and human tasks. It is exemplified with the help of a case study concerning a web-based application intended to manage the handling of paper submissions to conferences.

1 INTRODUCTION

At a simplified conceptual level, an enterprise information system (EIS) can be analyzed from two major perspectives: the informational perspective and the behavioral one. The informational perspective is concerned with the structure of information; the behavioral perspective, instead, addresses the organization of the units of work (called tasks) which enable the (human) users and the system itself to operate on the underlying informational base.

While the informational perspective is based on one kind of models, which show the classes of the information items (called business objects) and their relationships, for the behavioral perspective several kinds of models have been proposed with different purposes. Two of them, use cases and business processes, deserve particular attention.

UML use cases (OMG, 2007) are mainly used in the requirements specification phase of the software life cycle: they show the tasks and the roles required for them, and can also point out the structural dependencies (inclusion and extension) among the tasks. What is missing from use case models is the information flow, i.e. the indication of the information items to be operated on by the tasks; on the contrary, the information flow played an important role in the old functional models, such as the dataflow diagrams (Gane and Sarson, 1979).

Business processes can be addressed with different modeling notations, such as BPMN (OMG, 2006) and UML activity diagrams (OMG, 2007), however they all have a point in common: the emphasis placed on the control flow, on the basis of the well-known workflow patterns (van der Aalst, ter Hofstede, Kiepuszewski and Barros, 2003), and the omission of the information flow (although information items can be included for documentation purposes).

Moreover, the interactions between the process elements and the human tasks are understated. In fact, a business process is meant to be mainly an orchestrator of external activities. There are two main reasons for this: 1) the limited processing capabilities of the current languages for business processes, such as XPDL (WfMC, 2005); 2) the consideration that the information resides in the EIS and can be manipulated more effectively with enterprise operations invoked through services. The external activities are represented by the process elements (or process steps), which also indicate how to activate them. In particular, in BPMN, the connection between a process step and a task is obtained through an asynchronous service. The process step indicates the performer required (in terms of a single string or an array of strings), the input message, which is meant to activate the task and to provide it with the input information, and the output message, which is meant to notify the completion of the task as well as to return the output

Bruno G. (2007).
INTEGRATING BUSINESS PROCESSES AND INFORMATION SYSTEMS.
In *Proceedings of the Second International Conference on Software and Data Technologies - Volume ISDM/WsEHST/DC*, pages 79-86
DOI: 10.5220/0001327200790086

information. This simple interaction pattern does not address all the situations taking place in practical applications. For example, if task “submitPapers” is meant to enable an author to submit a number of papers over a given period of time and to make each paper immediately processed (i.e. assigned to some reviewers), then this task will generate a number of output messages, one for each paper submitted, not only the completion one.

The selection of the performer for a human task is a critical issue and several patterns have been proposed (Russell, van der Aalst, ter Hofstede, and Edmond, 2005). In some cases, the performer may be selected among the users playing a given role, with a load-balancing criterion. In other cases, the selection may depend on the information flow: for example, the performer of task “reviewPaper” is the reviewer that has previously been associated with the paper to be reviewed.

This paper presents a notation for business processes whose purpose is to integrate the control flow and the information one so as to provide a better integration with human tasks. This notation, called tk-nets (task-oriented nets), has a structure similar to Petri nets, however the interpretation is tailored to that purpose.

The organization of this paper is as follows. Section 2 presents four interaction patterns between process steps and human tasks. Section 3 introduces a case study concerning a web-based application intended to manage the handling of paper submissions to conferences: it stresses the need for a tight integration between the control flow and the information one. Section 4 presents the tk-nets notation exemplified on the case study. Section 5 provides a comparison with related work. Section 6 presents the conclusion and the future work.

2 INTERACTION PATTERNS BETWEEN PROCESS STEPS AND TASKS

A (human) task is a unit of work that a suitable user (or actor) carries out with the help of a graphical user interface (GUI) in order to achieve a particular purpose. Placing a purchase order or filling in the review form for a conference paper are examples of tasks. As such, a task is an abstract entity: in reality, a task encompasses both foreground actions, i.e. the user entering pieces of information or commands, and background actions, i.e. the system processing the user’s inputs. Such background actions rely on enterprise operations in order to operate on the underlying business objects.

Tasks can be activated in two modes, denoted by terms “pull” and “push”.

In the “pull” mode, tasks are referred to by the menu items available in the GUI, according to the role(s) played by the current user. A menu item, such as “review paper”, denotes a potential task: if the user clicks it, the system shows (either directly or through a search) the actual instances of that task the user is expected to perform (e.g. the papers to review). In other terms, the user is pulling the system. An actual instance of a task, such as “review paper n. 123”, is called an actual task.

In the “push” mode, users are presented with a to-do list showing the actual tasks that have been assigned to them; by clicking an item of the list, a user can activate the corresponding task.

The purpose of business processes is to make users work in the “push” mode: in fact, the control flow determines when a task has to be performed and the information flow indicates the information needed.

From an operational point of view, the bridge between processes and tasks is provided by the to-do list, which is based on a particular class of business objects, called task assignments. A task assignment basically designates the user in charge of the task (the performer), the input information and the timing constraints (the period in which the work has to be performed). A task assignment goes through a number of states, the major of which are “assigned” (the initial state), “enabled”, “started” and “ended”. When a task assignment is in state “enabled”, it can be shown in the to-do list of the corresponding performer. When the user begins working on the actual task, the state of the task assignment becomes “started”. A task is a long-running activity, as the performer can stop and resume work several times, until (s)he states it is ended.

To-do lists are handled by a specific architectural component, called the task manager (TM). It is a kind of mediator between the process manager (PM) in charge of interpreting the descriptions of business processes and the enterprise operations supporting the GUI.

While processes are expected to assign tasks to performers, current notations and languages do not provide direct references to tasks, but only indirect ones through intermediate services. In fact, a business process, as described in BPMN or XPDL, consists of a number of elements, or process steps, and the connection between a process step and a task is obtained through an asynchronous service: the input message (with respect to the service) is meant to activate the task and to provide it with the input information, while the output message is meant to notify the completion of the task as well as to return the output information.

This simple interaction pattern does not cover all the situations taking place in practical applications. In terms of events, it addresses the case of one input event (with respect to TM) and one output event: the input event entails the generation of a task assignment and the output event notifies the completion of the task.

This paper proposes three more patterns so as to cope with multiple events. The four patterns are called task interaction patterns. The above mentioned one is referred to as pattern (1, 1).

Pattern (1, *) indicates that the task (i.e. the related enterprise operations) sends a number of intermediate events before the completion one. Such additional output events signify that the task has produced information items requiring immediate attention. For example, if task “submitPapers” is meant to enable an author to submit a number of papers over a given period of time and to make each paper immediately processed (i.e. assigned to some reviewers), then this task will generate a number of intermediate events, one for each paper submitted.

Pattern (*, 1) indicates that the task receives additional input events, after the activation one, signifying that additional input information is available. For example, task “assignPapers” is activated with an initial group of papers and an initial group of reviewers and then, during its execution, it receives additional papers and reviewers: this way, the conference chair is able to continuously monitor the match between papers and reviewers and, if needed, (s)he has time to involve more reviewers.

The fourth pattern (*, *) denotes both a flow of input events and a flow of output ones. For example, task “evaluatePapers” is meant to receive a flow of reviews and to provide a flow of papers evaluated, each of which will trigger an immediate notification to the author.

3 THE CASE STUDY

The case study addresses a fictitious web-based application intended to manage the submissions of papers to conferences.

In particular, attention is focused on process ConferenceBP aimed at handling the life cycle of a given conference. It is a case-based process, in the sense that a particular instance of this process handles the life cycle of a specific conference (i.e. a case).

A conference is represented by a business object of class Conference. The simplified class-model for the business objects supporting this application is shown in Figure 1 (the default multiplicity value for association ends is *, i.e. many).

The users of the system belong to three major roles: (conference) chair, author and reviewer; they are represented by business objects of classes Chair, Author and Reviewer, respectively. When the process is started, the general information about the conference (including the submission period, the review period and the registration period) has already been established; moreover the conference business object has already been associated with the chair business object.

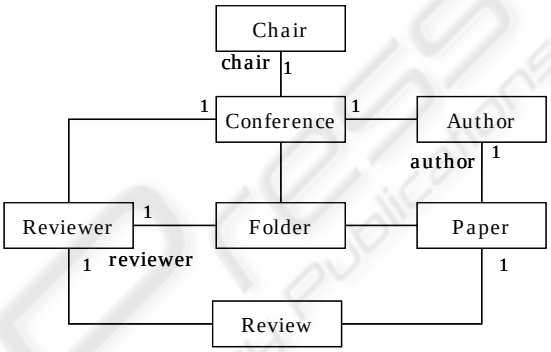

Figure 1: Class model related to the case study.

The behavior of ConferenceBP is as follows. During the submission period, users can get an account as authors or reviewers. Authors can then submit papers (an author can submit several papers). As papers are submitted and reviewers join the conference, the conference chair goes on assigning papers to reviewers: in particular, all the papers assigned to a reviewer are collected in a folder. When the chair has finished, the reviewers can start working on their folders: they are assumed to release their reviews separately during the review period. As reviews are released, the chair goes on evaluating papers: if it happens that the number of reviews of a given paper is not sufficient to take a decision, the chair can involve other reviewers by providing them with new folders containing the pending papers. When all the decisions have been taken, a notification is emailed for each paper and the conference rate is automatically determined for each author. Then, during the registration period, the authors of accepted papers submit the final versions and in parallel they also enter payment information details. As final papers are submitted and payment information is provided, the chair goes on defining the conference program.

The requirements above indicate a number of tasks which can be grouped on the basis of their interaction patterns, as follows.

(1, 1): getAuthorAccount, getReviewerAccount, submitFinalPaper, enterPaymentInfo.

(1, *): submitPaper, reviewPapers.

(*, 1): assignPapers, reassignPapers, evaluate-Papers, defineProgram.

4 MODELING BUSINESS PROCESSES WITH TK-NETS

The model of ConferenceBP is shown in Figure 2. The notation proposed in this paper, i.e. tk-nets, is illustrated in this section.

Process ConferenceBP is a case-based process describing the life cycle of a conference: this is indicated with annotation “manages Conference” in Figure 2, where Conference is the “managed” class, i.e. the class of the business objects (called managed objects) managed by the process instances.

There are two kinds of transitions (or process steps) in tk-nets: those related to tasks (referred to as task transitions) are represented as rectangles with rounded corners, while regular rectangles represent procedures that can automatically be performed by the system. Each transition is mapped to a process action describing its detailed behavior. An informal definition of the process actions associated with ConferenceBP is presented in Table 1. The role of process actions will be discussed in subsection 4.1.

A task transition indicates the task involved, the interaction pattern and the role of the performer: in fact, the name of the task is the name of the task transition, the interaction pattern is indicated by the stereotype (e.g. <<1, *>>), and the role required is written between parentheses. The action of a task transition can give rise to a number of similar tasks; in that case, for documentation purposes, the task transition (e.g. “reviewPapers”) is shown shaded.

There are two kinds of places in tk-nets, i.e. typed places and simple ones. Simple places are depicted as small grey circles, such as “c1” and “c2”; they only have a name. Typed places are depicted as larger circles and have a label consisting of the place name, followed by the place type.

The process elements, places and transitions, are formal items, in the sense that they are a kind of templates for the actual items belonging to the process instances. In fact, a process instance is made up of actual places, each actual place referring to the corresponding formal place. Actual places contain tokens: typed places contain typed tokens, while simple ones contain simple (or empty) tokens. A typed token is associated with a business object; the class of the business object coincides with the type of the corresponding formal place. A typed token represents a state of the related business object; for example, a token in place “p1” denotes a paper that has just been submitted, in the context of a certain instance of ConferenceBP.

The information flow and the control flow may overlap at typed places, and, in order to explain what it means, the notion of triggers is introduced.

From a logical point of view, a trigger is issued by an actual place in three modes, as follows. If the place is a fully triggering (or “ft”) place, it issues a trigger as soon as it receives a token; if it is a non-triggering (or “nt”) place, it never issues triggers; if it is a partially triggering (or “pt”) place, it issues a trigger only when an input transition ends. The “pt” behavior is based on a special event, i.e. the ending of an input (with respect to the place) transition. A procedure ends when the corresponding action has been performed, while a task transition ends when all the tasks it has activated are completed. This way, a task transition, besides activating a number of tasks, can synchronize their completion.

The purpose of triggers is as follows. When a trigger is issued by an actual place, say p, if the corresponding formal place has just one outgoing transition, the process manager (PM) tries to perform that transition by calling the associated process action. A process action incorporates a guard, which can accept or reject the trigger on the basis of the tokens available in the actual input places, as will be discussed in subsection 4.1. In case the formal place has two or more outgoing transitions, PM call the corresponding actions in sequence, on the basis of their priorities, until the guard of one of them is successful. In Figure 2, the priority is the same for all the transitions.

Simple places are always “ft” places: in fact, they support the control flow only. The triggering behavior of typed places is indicated by acronym “ft”, “nt” or “pt” depicted in the circle. A graphical alternative is adopted in Figure 2: the white typed places denote “ft” places, and the grey typed places denote “pt” places; there are no “nt” places.

The information flow of a transition is indicated by its surrounding typed places. In order to make the model more expressive, the input arcs and the output ones of task transitions are shown thin or thick. A thin input (with respect to a transition) arc means that the task will receive a single token from the corresponding input place. Likewise, a thin output arc means that the task will deliver a single token to the corresponding output place. In case of thick arcs, the task will receive or deliver a number of tokens, instead of a single one. For example, the outgoing arc of “submitPaper” is thick, as an author can submit several papers.

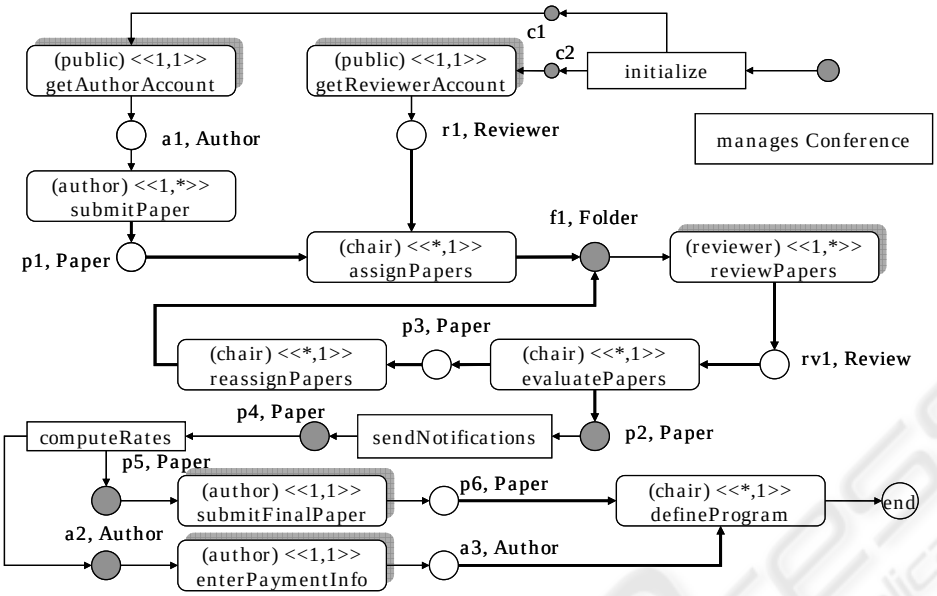

Figure 2: The tk-nets model of process ConferenceBP.

Table1: Actions of process ConferenceBP.

getAuthorAccount: < → "public">;
getReviewerAccount: < → "public">;
submitPaper: < → a1>;
assignPapers: <p1 or r1 → conference.chair>;
reviewPapers: for each f1 in f1* {<f1 → f1.reviewer>};
evaluatePapers: <rv1 → conference.chair>;
reassignPapers: <p3 → conference.chair>;
submitFinalPaper: for each p5 in p5* {<p5 → p5.author>};
enterPaymentInfo: for each a2 in a2* {<a2 → a2>};
defineProgram:<p6 or a3 → conference.chair>;
sendNotifications: for each p2 in p2* send acceptance message or rejection message according to p2 state (i.e. accepted or rejected); deliver only the accepted papers to place p4;
computeRates: compute the rates for the authors of the papers accepted (those in p4*); deliver papers to place p5 and authors to place a2;

4.1 Description of ConferenceBP

This subsection describes the behavior of the instances of ConferenceBP as well as the role of the process actions.

A process action is called by PM when an input place has issued a trigger. An action incorporates a guard, whose purpose is to check whether all the tokens required are present in the input places. If the guard is successful, the trigger is accepted and the action takes such tokens (called the input tokens) from the input places and operates on the business objects associated with them. If the guard fails, the action is not performed. If the guard is omitted, a standard behavior is assumed, as follows: the trigger is always accepted and the action automatically takes the first token from the "ft" places and all the available tokens from the "pt" places and the "nt" ones.

Within the body of an action, the business objects associated with the input tokens are referred to with the names of the places: in particular, if one token is taken from place p, the corresponding business object is referred to as p, while, if several tokens are taken from place p, the collection of the business objects associated with them is referred to as p*. The business objects associated with the input tokens are also called the input business objects.

Task transitions are meant to produce task assignments by means of task requests. A task request basically designates the user in charge of the task (the performer), the input information and the timing constraints (the period in which the work has to be performed). Task requests are shown, in the actions, in the simplified form <i → p>, where i denotes the input information, i.e. the input business objects, and p denotes the performer. The managed object, i.e. the current conference in this case, is assumed to be implicitly part of the input information. If the task is meant to receive a flow of input information, i.e. its interaction pattern is <<*, 1>> or <<*, *>>, the effect of the task request is as follows: if there is an ongoing task assignment, the input information is added to it, otherwise a new

task assignment is generated and the input information is associated with it.

All the transitions in ConferenceBP have a standard behavior, hence the guards are omitted.

When an instance of ConferenceBP is started, procedure "initialize" is performed, and then the actions of both "getAuthorAccount" and "getReviewerAccount", which are similar, are carried out.

In the action of "getAuthorAccount", task request < → "public"> indicates that there is no specific input information, apart from the current conference. Moreover, the actual performer is not specified; only a role name ("public") is provided. In fact, this task is not intended for a known user (i.e. a user registered in the EIS): in the web page related to the conference, there will be a menu item (e.g. "getAuthorAccount") enabling any interested user to get an author account.

Task transition "getAuthorAccount" gives rise to a number of actual tasks; in fact, as long as the submission period is open, new authors can register. However, as soon as an author has registered, task "submitPaper" is assigned to them; for this reason, place "a1" is a fully triggering place (i.e. a grey place).

In the action of "submitPaper", task request < → a1> indicates that the performer is the user (an author) denoted by business object "a1", i.e. the author who has just registered.

Task "assignPapers" features interaction pattern (*, 1). In fact, it is meant: a) to receive a flow of input information (papers and reviewers); b) to deliver a number of folders to place "f1" (as shown by the thick arc), but only one output event (i.e. a trigger) to "f1", when it ends (hence "f1" is a "pt" place). Task request <p1 or r1 → conference.chair> indicates that the performer is the conference chair: in fact, "conference.chair" is a kind of navigational construct which returns the chair (business object) associated with the current conference on the basis of its relational attribute "chair" (shown in Figure 1). The input information consists of one paper or one reviewer (i.e. the corresponding business objects), as both places "p1" and "r1" send separate triggers.

When task transition "assignPapers" ends, task transition "reviewPapers" is performed. Its action contains a number of task requests, one for each folder taken from place "f1". Each task "reviewPapers" is assumed to deliver a number of reviews (as reviewers are not supposed to do their work all at once) which are passed to task "evaluatePapers". This task can deliver two output flows: the first flow, made up of the papers for which a decision has been taken, is directed to place "p2", while the second one, consisting of the pending papers, is directed to place "p3". As the end of the review period is approaching, the chair can deliver the pending papers to place "p3" thus activating task transition "reassignPapers" by means of which (s)he can prepare new folders for other reviewers. When all the papers have been evaluated, task "evaluatePapers" ends and triggers procedure "sendNotifications". The behavior of the remaining transitions is described in Table 1.

5 COMPARISON WITH RELATED WORK

The various notations proposed for business processes can be compared from different points of view, such as the control-flow perspective (van der Aalst, ter Hofstede, Kiepuszewski and Barros, 2003).

This section discusses how the control flow and the information flow are handled in three major notations, i.e. the Event-driven Process Chain, Colored Petri Nets, and BPMN.

The Event-driven Process Chain (EPC) is an informal notation which has been used to describe the SAP R/3 reference model (Curran and Keller, 1998). It is based on the notions of event, function, information item and organization unit. The control-flow structure is based on events, functions and control-flow links connecting events and functions, while the information-flow structure is based on information items, functions and information-flow links connecting information items and functions. Recent work (Mendling, Neumann and Nüttgens, 2005) has been aimed at formalizing the control flow semantics under the pressure of the workflow patterns (van der Aalst, ter Hofstede, Kiepuszewski and Barros, 2003). However, the separation between the control-flow links and the information-flow links entails some redundancy.

In fact, if two functions operate in series, as the information item produced by the first one is acted on by the second one, then an intermediate event is needed to propagate control from the first function to the second one, in addition to the information item.

In Colored Petri Nets (CPNs) (Kristensen, Christensen and Jensen, 1998), the control-flow structure and the information-flow one coincide: in fact, transitions are token-driven processing units and tokens carry pieces of information. As CPNs have not been designed to specifically address human tasks, transitions can only represent the first of the task interaction patterns presented in section 2, and therefore places in CPNs correspond to only one category of places in tk-nets, i.e. the fully-triggering places.

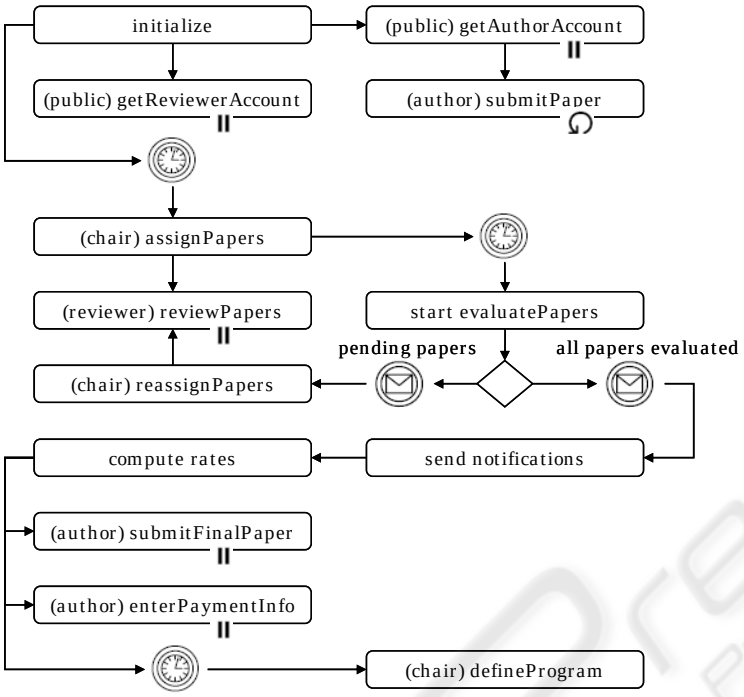

Figure 3: The BPMN model of ConferenceBP.

BPMN (OMG, 2006) and UML activity diagrams (OMG, 2007) provide the control flow, but not the information flow (although for documentation purposes information items can be included).

In Figure 3, the model of ConferenceBP is presented with the BPMN notation. The major difficulty lies in representing human tasks (e.g. “evaluatePapers”) that can receive and/or deliver a flow of information, as will be discussed later on in this subsection.

Process steps “getAuthorAccount” and “getReviewerAccount” feature the multiple-instance property (indicated by the parallel mark). In fact, they activate a number (not known a priori) of the corresponding human tasks. The BPMN MI_FlowCondition attribute of “getAuthorAccount” can be set to a particular value so as to produce an output token at the end of each instance (i.e. when a user has got an account): this way, step “submitPaper” can be activated. The loop marker in “submitPaper” indicates that this step can be repeated; in fact, an author can submit a number of papers.

Since task “assignPapers” is not strongly related to the end of the previous tasks, it is activated after a certain period of time after the “initialize” step, as shown by the delay step represented by the watch symbol. The information flow is not represented, therefore it is up to the implementation of the task to collect the business objects related to the papers submitted and to the reviewers registered. When this task is completed, the step ends and releases two output tokens, one triggering the multiple-instance “reviewPapers” step, and the other activating step “start evaluatePapers”, after a certain delay. Actually, step “start evaluatePapers” does not coincide with human task evaluatePapers, because this task may emit intermediate events in case of (pending) papers to be reassigned to other reviewers.

A task issuing multiple events cannot be represented in BPMN with a single process step. The solution adopted in Figure 3 is to start the task through a service (in step “start evaluatePapers”) and then wait for one of two events to occur: in case of event “pending papers”, step “reassignPapers” is performed, while in case of event “all papers evaluated”, step “sendNotifications” is carried out.

The analysis of workflow data patterns (Russell, ter Hofstede, Edmond and van der Aalst, 2005) mainly addresses the various ways in which data items are represented and handled within a business process; in tk-nets, however, data items are external as they are part of the EIS.

6 CONCLUSION

While it is advocated (Dumas, van der Aalst, and ter Hofstede, 2005) that EISs be aware of business processes, the vice versa is no less true.

This paper has proposed an approach aimed at strengthening the relations between business processes and EISs by making the processes aware of the human tasks and the information flow. This approach is based on a notation called tk-nets, which is complemented by a procedural part, i.e. the process actions, describing the detailed behavior of transitions. A prototype of the interpreter for tk-nets has already been implemented: current work is being devoted to making the approach fully operational: for this reason, process actions are mapped to enterprise beans according to the Java EE standard, and, moreover, simulation features for human tasks are under development.

Another direction of research is concerned with the use of tk-nets to represent business models on the basis of different perspectives, such as the role-activity perspective (Ould, 2005) and the language/action one (Winograd, 1987-88).

REFERENCES

Curran, T., Keller, G., 1998. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Prentice Hall.

Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., 2005. *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley.

Gane, C.P., Sarson, T., 1979. *Structured Systems Analysis: Tools and Techniques*. Prentice Hall.

Kristensen, L.M., Christensen, S., Jensen, K., 1998. The Practitioner's Guide to Coloured Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2, 98-132.

Mendling J., Neumann, G., Nüttgens, M., 2005. Yet Another Event-Driven Process Chain. In *Lecture Notes in Computer Science*, 3649, 428–433, Springer. Berlin.

OMG, 2006. Business Process Modeling Notation (BPMN), Final Adopted Specification, February 2006. Retrieved March, 10, 2007, from http: /www.bpmn.org/.

OMG, 2007. Unified Modeling Language: Superstructure, version 2.1.1. Retrieved March, 10, 2007, from http://www.omg.org/docs/formal/07-02-03.pdf.

Ould, M., 2005. *Business Process Management: A Rigorous Approach*. The British Computer Society.

Russell, N., ter Hofstede, A.H.M, Edmond, D., van der Aalst, W.M.P, 2005. Workflow Data Patterns: Identification, Representation and Tool Support. In *Lecture Notes in Computer Science*, 3716, 353-368, Springer. Berlin.

Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D., 2005. Workflow Resource Patterns: Identification, Representation and Tool Support. In *Lecture Notes in Computer Science*, 3520, 216–232, Springer. Berlin.

van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P., 2003. Workflow Patterns. *Distributed and Parallel Databases*, 14, 5-51.

WfMC, 2005. XML Process Definition Language, Version 2.00. Retrieved March, 10, 2007, from http://www.wfmc.org.

Winograd, T., 1987-1988. A Language/Action Perspective on the Design of Cooperative Work. *Human-Computer Interaction*, 3, 3-30.