UNIVERSITY OF SÃO PAULO

INSTITUTE OF MATHEMATICS AND STATISTICS

DEPARTAMENT OF COMPUTER SCIENCE

Technical Report RT-MAC-2011-03

FROM BUSINESS PROCESS MODEL AND NOTATION TO STOCHASTIC AUTOMATA NETWORK

Kelly Rosa Braghetto, João Eduardo Ferreira and Jean-Marc Vincent

March, 2011

From Business Process Model and Notation to Stochastic Automata Network

Kelly Rosa Braghetto[1] * **, João Eduardo Ferreira[1], and Jean-Marc Vincent[2]

[1] Department of Computer Science, University of São Paulo
Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, Brasil
{kellyrb,jef}@ime.usp.br
[2] LIG Laboratory – INRIA MESCAL Project, Joseph Fourier University
51, avenue Jean Kuntzmann, F-38330, Montbonnot, France
Jean-Marc.Vincent@imag.fr

Abstract. The qualitative and quantitative analysis of operational processes recently started to receive special attention with the *business process management* systems. But the *Business Process Model and Notation* (BPMN), the standard representation of business processes, is not appropriate to support the analysis phase. Most of the works proposing mappings from BPMN to formal languages aim model verification, but few are directed to quantitative analysis. In this work, we state that a well-defined BPMN Process diagram can originate a *Stochastic Automata Network* (SAN) – a compositionally built stochastic model. More than support verification, SAN provides a numerical evaluation of processes' performance. SAN attenuates the state-space explosion problem associated with other Markovian formalisms and is used to model large/complex systems. The main contribution of this work is an algorithm that converts BPMN diagrams to SAN. This conversion is the first step to build complete performance evaluation models of business processes.

Keywords: Business Processes, BPMN, Performance Evaluation, Stochastic Automata Network

1 Introduction

The *business processes management* (BPM) systems can be seen as a "natural evolution" of the *workflow management* (WFM) systems. When the WFM systems first appeared, in the 80's, their main target was process automation. But with the evolution of the specific-domain modeling languages and supporting tools, the analysis (both qualitative and quantitative) of the operational processes – almost unattended in the WFM systems – recently started to receive special attention with the BPM systems.

Several efforts have been made to standardize modeling and execution languages, in order to improve the interoperability of tools developed to BPM. The

* The student was supported by the Brazilian government (CAPES and FAPESP).
** Contact author. The order of authors is merely alphabetical.

most important result of these efforts is the *Business Process Model and Notation* (BPMN) [12], a standard notation for graphical representation of business processes. Despite being able to support business users in different phases of the business process life cycle, BPMN models are not appropriate to support the analysis phase. Since BPMN models have no formal semantics, they are not well suited to qualitative analysis (validation and verification). Furthermore, BPMN diagrams do not provide mechanisms to quantify the computational/human effort required to perform the activities, nor the capacity of work of shared resources and their access policies. This deficiency hinders the use of BPMN for performance evaluation (i.e., quantitative analysis). In BPM domain, the objective of performance analysis is to evaluate performance indicators – service time, waiting time, queue size, and resource utilization – that enable us to improve the business processes by identifying inefficiencies, such as bottlenecks and idle resources.

Several recent works proposed mappings from BPMN to formal languages, in order to enable model verification. But few works regard business process modeling aiming performance evaluation. The analytical modeling for performance evaluation is generally made over stochastic models that have as underlying formalism a Markov process. This kind of models are used to represent uncertainty in systems that evolve dynamically in time. BPMN models are easily readable, their syntax and semantics are intuitive even for non technical users. On the other side, reading or designing stochastic models is far from being a trivial task, it demands some statistical skill in addition to the familiarity with the formalism.

In this work, we state that a well-defined BPMN Process diagram can originate a *Stochastic Automata Network* (SAN) [14]. More than support verification, SAN models are able to provide a numerical performance analysis of business processes. SAN is a structured Markovian formalism that enables us to build stochastic models in a compositional approach. Created to attenuate the well-known state-space explosion problem associated with the Markovian formalisms, SAN can be applied in the modeling of large/complex systems. It is very efficient regarding the memory consumption, in addition to provide the concept of functional rates – that can facilitate the modeling and help to reduce the size of the state-space.

Our main contribution is an algorithm that automatically converts BPMN diagrams to SAN models, by means of a set of mappings and simple operations that we defined over the models. This conversion is the first step required to build the performance evaluation model of a business process. After, the SAN model generated by the conversion algorithm should be enriched with information regarding the resource management associated with the modeled business process. This complementary step generates a model that, when numerically analyzed, provides variated performance indicators of the business process.

The paper's remainder is organized as follows. Section 2 introduces the two topics required to the understanding of this work: the SAN formalism and the BPMN notation. Our algorithm for the conversion of BPMN models to SAN models is presented and formalized by definitions in Section 3; the section also provides a short example of conversion. Section 4 discusses some related works, while the concluding remarks are made in Section 5.

2 Fundamentals

Two topics are indispensable for the understanding of this work: the SAN formalism and the main structures of the BPMN Process diagrams. We briefly introduce them in this section.

2.1 Stochastic Automata Network

The *Stochastic Automata Network* (SAN) is a technique used to model systems with large state spaces, introduced by Plateau in 1985 [14, 15]. SAN has been successfully applied to model parallel and distributed systems that can be viewed as collections of components that operate more or less independently, requiring only infrequent interaction such as synchronizing their actions, or operating at different rates depending on the state of parts of the overall system.

A system is described in SAN as a set of N subsystems modeled as a set of stochastic automata $A^{(i)}$, $1 \leq i \leq N$, each one containing n_i local states and transitions among them. The global state of a SAN is defined by the combinations of the internal state of each automaton. A change in the state of a SAN is caused by the occurrence of an *event*. *Local events* cause a state transition in only one automaton (*local transition*), while *synchronizing events* cause simultaneous state transitions in more than one automaton (*synchronizing transitions*). A transition is labeled with the list of events that may trigger it.

All event transitions in the model are associated to rates, that indicates the average frequency (or, in other words, the inverse of the average execution time) in which the transitions occur. The rate of an event may be constant (a nonnegative real number) or may depend upon the state in which it takes place. In this last case, the rate is a function from the global state space to the nonnegative real numbers and is called *functional transition rate*. This concept of functional rate of SAN (where the rate can depend on the entire state of the SAN model) is more general than the concept of *state dependent service rate* existing in queuing networks (where the rate depends only on the state of the queue itself). For example, one can use functional transition rate to model how the execution time of an activity in the system is affected by the variation of the workload, or to model dependency existent between the probability of the execution of an activity and the current state of the process.

The expression of the *infinitesimal generator* (transition rate matrix) of the underlying Markov chain of a well defined SAN is given by the generators on these smaller spaces and by operators from the *Generalized Tensor Algebra* (GTA) [6], an extension of the *Classical Tensor Algebra* (CTA). The tensor formula that gives the infinitesimal generator of a SAN model is called *Markovian Descriptor*.

Each automaton $A^{(i)}$ of a SAN model is described by $n_i \times n_i$ square matrices. In the case of SAN models with synchronizing events, the descriptor is expressed in two parts: a *local* part (to group the local events), and a *synchronizing* part (to group the synchronizing events). The local part is defined by the tensor sum of $Q_l^{(i)}$ – the infinitesimal generator matrices of the local transitions of each $A^{(i)}$.

In the synchronizing part, each event corresponds to two tensor products: one for the occurrence matrices $Q_{s^+}^{(i)}$ (expressing the positive rates) and the other for the adjusting matrices $Q_{s^-}^{(i)}$ (expressing the negative rates). The descriptor is the sum of the local and the synchronizing parts, expressed as:

$$Q = \bigoplus_{i=1}^{N}{}_g \, Q_l^{(i)} + \sum_{s \in \varepsilon} \left(\bigotimes_{i=1}^{N}{}_g \, Q_{s^+}^{(i)} + \bigotimes_{i=1}^{N}{}_g \, Q_{s^-}^{(i)} \right) \tag{1}$$

where $\begin{cases} N \text{ is the number of automata of the SAN model} \\ \varepsilon \text{ is the set of identifiers of synchronizing events} \end{cases}$.

The state-space explosion problem associated with Markov chain models is attenuated by the fact that the state transition matrix is stored in a compact form, since it is represented by smaller matrices. All relevant information can be recovered from these matrices without explicitly build the global matrix.

The *steady state analysis* of a SAN model gives the stationary probability distribution of the system – i.e., the long-run average time the system spends in each one of its states. From this stationary distribution we are able to extract performance indices such as average throughput of the system, average waiting time, average queues size, utilization rate of resources, etc. Contrarily to the steady state analysis that is interested in the long-run behavior, the *transient analysis* investigates the transient behavior of the process, i.e., it is able to determine the state of the process at the end of a time interval, the time until an event occurs, the number of occurrences of an event during a time interval, etc.

A SAN model can be numerically solved using the PEPS tool [5][3]. PEPS includes several numerical iterative methods to solve SAN models and implements strategies to improve the time/space trade-off in the computation of the solutions. It supports both steady state analysis and transient analysis.

2.2 Business Process Model and Notation

The *Business Process Model and Notation* (BPMN) is a standard maintained by the *Object Management Group* (OMG). The BPMN's main goal is to work as "a standardized bridge for the gap between the business process design and process implementation" [12]. The intention was to provide a common notation for the business users involved in different phases of the life cycle of a business process.

Using BPMN, we are able to build three types of diagrams: *Collaboration* diagrams, *Process* diagrams, and *Choreography* diagrams. Since the management of resources can greatly vary from one type of diagram to other, in this work we restricted ourselves to the Process diagrams. According to the specification document of BPMN, "a process describes a sequence or flow of activities in an organization with the objective of carrying out work". A Process diagram is a

[3] The PEPS tool and other details about it are available in `http://www-id.imag.fr/Logiciels/peps/index.html`.

graph of elements – activities, events, gateways, and sequence flows – that define a finite execution semantics.

In this work we deal with a subclass of the models that can be represented as a BPMN Process diagram – the well-defined ones (Section 3.1, Definition 5) – , in order to guarantee that the conversion of the diagram to a SAN model can be made. Table 1 introduces the BPMN objects accepted as input by our conversion algorithm. These objects are very important to process modeling and with them we are able to express a rich class of business process models. In the following, we briefly describe these graphical objects according to the definitions of the specification document.

Start Event	Exclusive Gateway	Sequence Flow
End Event	Parallel Gateway	Association
Atomic activity	Inclusive Gateway	Annotation

Table 1. Basic flow and connecting objects of BPMN.

An *event* is something that happens during the course of a process and affects the flow of the model. The *start event* indicates where a process will start, and *end event* indicates where a process will end. An *activity* is a generic term for work performed in the process; it can be atomic or compound. In this work, the term *activity* refers to an atomic activity. A *sequence flow* is used to show the order in which activities will be performed. A *gateway* is used to control the divergence and convergence of sequence flows. Gateways can have several behavior controls and each type of control affects both the incoming and outgoing flow. In this work, we deals with the following gateway types:

- *Exclusive Gateways.* A *diverging exclusive gateway* (decision) is used to create alternative paths within a process flow. For a given instance of the process, only one of the paths can be taken. A *converging exclusive gateway* is used to merge alternative paths. Each incoming control flow token is routed to the outgoing sequence flow without synchronization;
- *Parallel Gateways.* A *diverging parallel gateway* creates parallel paths without checking any conditions. The *converging parallel gateway* will wait for all incoming flows before triggering the flow through its outgoing sequence flows;
- *Inclusive Gateways.* A *diverging inclusive gateway* can be used to create alternative but also parallel paths within a process flow. Unlike the exclusive gateway, all condition expressions are evaluated. All sequence flow with a true evaluation will be taken. A *converging inclusive gateway* is used to merge a combination of alternative and parallel paths; an arriving sequence flow may be synchronized with some other sequence flows that arrive later.

In a parallel between BPMN objects and the workflow terminology, an exclusive gateway corresponds to a XOR-split/join, a parallel gateway corresponds to an AND-split/join, and an inclusive gateway corresponds to an OR-split/join.

An *association* is used to link information with graphical elements. *Text annotations* provide additional information for readers of the BPMN diagrams.

Other BPMN objects can be expressed in therms of the objects of Table 1. For example, *activity looping* and *multi-instances activities* can be modeled using atomic activities and exclusive/parallel gateways.

Process Instantiation and Completion. A process is instantiated when one of its start events occurs. But, before introducing the notion of *process completion*, we need to present the concept of *token* in BPMN.

A token is a theoretical concept that is used in the BPMN specification [12] document as an aid to informally define the operational semantics of the BPMN constructions. The behavior of process elements can be defined by describing how they interact with the token as it traverses the structure of the process.

Each start event that occurs creates a token on its outgoing sequence flow, which is followed as described by the semantics of the other process elements. For example, the parallel gateway is activated if there is at least one token on each incoming sequence flow; the parallel gateway consumes exactly one token from each incoming sequence flow and produces exactly one token at each outgoing sequence flow. Having this in mind, we can consider that a process instance is completed if and only if the following two conditions hold: (i) there is no token remaining within the process instance; (ii) no activity of the process is still active.

3 Conversion of BPMN Graphs to SAN Models

Section 3.1 introduces the *structural* properties of a BPMN graph that we consider as a valid input for our conversion algorithm. In Section 3.2 we formally define the structure of the SAN graph resulted from a conversion, and the operations over SAN graphs that support the algorithm (described in Section 3.3).

3.1 BPMN Graph Definitions

As we briefly discussed in Section 2.2, a BPMN Process diagram is a directed graph constituted of vertices of events, activities and gateways. In this work, we restricted ourselves to a subclass of all process diagrams that can be formed from BPMN objects. Definitions 1 to 5 formally describe this subclass.

Definition 1. *BPMN Process Graph*

A BPMN Process graph *BG is a directed graph represented by the tuple* $BG = (V, E, L, \ell, \mathrm{p})$*, with* $V = S \cup A \cup G \cup F$*, where:*

- S *is a set of vertices representing the* start events
- A *is a set of vertices representing the* atomic activities
- G *is a set of vertices representing the* gateways
- F *is a set of vertices representing the* end events
- $E \subseteq (V \times V)$ *is a set of* directed edges

- L *is a set of* vertex labels
- $\ell : V \longrightarrow L$ *is a* labeling function *of vertices*
- $\mathrm{p} : E' \longrightarrow [0,1]$, *where* $E' \subseteq E$, *is a* partial probability function *that associates edges with probability values*

A directed edge *in BG is a pair* (v, w) *– where* $v, w \in V$, *indicating that* v *is an input vertex of* w, *and* w *is an output vertex of* v. *A* label *is used to denote the name of the event or activity being modeled, but in the case of a gateway vertex, it indicates the gateway type.*

Definition 2. *Path in a BPMN Process Graph*

Let $BG = (V, E, L, \ell, \mathrm{p})$ *be a BPMN Process graph and* $v_1, v_n \in V$.

A path *from* v_1 *to* v_n *(represented by* $v_1 \rightsquigarrow v_n$*) is a sequence of vertices* $v_1, v_2, \cdots, v_n$ *(with* $v_i \in V$*) such that, for* $0 < i < n$, $(v_i, v_{i+1}) \in E$.

Definition 3. *Input Vertices and Output Vertices*

Let $BG = (V, E, L, \ell, \mathrm{p})$ *be a BPMN Process graph.*

The functions $\mathrm{inputs} : V \to 2^V$ *and* $\mathrm{outputs} : V \to 2^V$ *that give the* input vertices *and the* output vertices*, respectively, of a vertex in BP are defined as*

$$\forall v \in V, \mathrm{inputs}(v) = \{u \in V |\ (u, v) \in E\} \text{ and } \mathrm{outputs}(v) = \{w \in V |\ (v, w) \in E\}.$$

In order to minimize the semantic ambiguities that BPMN constructors can introduce in the model and to guarantee some properties that a well-formed business process model must respect, our conversion method makes some assumptions about the BPMN model provided as input (formalized in Definition 4). These assumptions facilitate the conversion and were already used (mainly the first 5 ones) in other related works such as, for example, in [16].

Definition 4. *Well-Formed BPMN Process Graph*

Let $BG = (V, E, L, \ell, \mathrm{p})$ *be a BPMN Process graph, with* $V = S \cup A \cup G \cup F$.

BG is a well-formed BPMN Process graph *if and only if:*

- $\forall v \in S$, $(|\,\mathrm{inputs}(v)| = 0) \wedge (|\,\mathrm{outputs}(v)| = 1)$
- $\forall v \in F$, $(|\,\mathrm{inputs}(v)| = 1) \wedge (|\,\mathrm{outputs}(v)| = 0)$
- $\forall v \in A$, $(|\,\mathrm{inputs}(v)| = 1) \wedge (|\,\mathrm{outputs}(v)| = 1)$
- $\forall v \in G$, $(\ell(v) = \text{“}+\text{”}) \vee (\ell(v) = \text{“}\bigcirc\text{”}) \vee (\ell(v) = \text{“}\times\text{”})$
- $\forall v \in G$, $((|\,\mathrm{inputs}(v)| > 1) \wedge (|\,\mathrm{outputs}(v)| = 1)) \vee ((|\,\mathrm{inputs}(v)| = 1) \wedge (|\,\mathrm{outputs}(v)| > 1))$
- $\forall v \in A \cup G$, $\exists s \in S$ *such that* $\exists$ *a path* $s \rightsquigarrow v$
- $\forall v \in A \cup G$, $\exists f \in F$ *such that* $\exists$ *a path* $v \rightsquigarrow f$
- $\forall v \in G$ *such as* $(\ell(v) = \text{“}\times\text{”}) \vee (\ell(v) = \text{“}\bigcirc\text{”})$, $\forall w \in \mathrm{outputs}(v)$, $\mathrm{p}((v, w))$ *must be defined*
- $\forall v \in G$ *such as* $\ell(v) = \text{“}\times\text{”} \sum_{w \,\in\, \mathrm{outputs}(v)} \mathrm{p}((v, w)) = 1$

Therefore, a well-formed BPMN Process graph is a graph in which:

- a start event vertex can have only one output vertex and no input vertices;

- an end event vertex can have only one input vertex and no output vertices;
- an activity vertex can have only one input vertex and only one output vertex;
- each gateway vertex has one of the following labels: " + ", for a parallel gateway; " ◯ ", for an inclusive gateway; and " × ", for an exclusive gateway;
- a gateway vertex can perform a role of divergence or a role of convergence (but not the two roles at the same time). As a consequence, a gateway can have only one input vertex and more output vertices (case of divergence), or can have only one output vertex and more input vertices (case of convergence);
- for all vertex v of activity or gateway: (i) there exists a path from one start event vertex to v, and (ii) there exists a path from v to one end event vertex;
- each output edge of an exclusive/inclusive gateway vertex must have an associated probability value;
- the sum of the probabilities of the output edges of an exclusive gateway vertex must be 1.

It is important to mention that the association of edges with probabilities does not exist in the specification of BPMN. We introduced this feature in our definition because the dynamic of a business process is probabilistic by essence and probabilities are quantifications of the business process behavior.

The assumptions above are all related to syntactical properties of the BPMN model, i.e. structural restrictions over the BPMN graph. But there exist also some important semantical properties that we need to assume to have a well-defined BPMN Process model (specified by Definition 5). In a well-defined BPMN Process model, two important properties for a business process are granted: (i) it does not contain unreachable activities, and (ii) it can always terminate.

Definition 5. *Well-Defined BPMN Process Model*
A well-defined BPMN Process model *is a well-formed BPMN Process graph in which:*

- *an exclusive gateway does not converge (join) parallel sequence flows;*
- *a parallel gateway does not converge (synchronize) alternative sequence flows;*
- *an inclusive gateway only converges (merges) sequence flows originated by another inclusive gateway. In addition, there is an one-to-one correspondence between the diverging and the converging inclusive gateways.*

3.2 SAN Model Definitions

Definitions from 6 to 11 formally specify the structure of a SAN model and its operations such as they are used in our conversion algorithm.

Definition 6. *SAN Model and SAN Automaton*
A SAN model $\mathcal{S}$ *is a set* $\mathcal{S} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$ *of* n *SAN automata.*
A SAN automaton $\mathcal{A}$ *is given by a tuple* $\mathcal{A} = (Q, E, T, L, \ell, \mathrm{p})$*, where:*

- Q *is a set of* states
- E *is a set of* events

- $T \subseteq (Q \times Q \times E)$ *is a set of* state transitions labeled by events
- L *is a set of* state labels
- $\ell : Q \longrightarrow L$ *is a* labeling function *of states*
- $\mathrm{p} : T' \longrightarrow [0, 1]$*, where* $T' \subseteq T$*, is a* partial probability function *that associates a transition with a probability value*

Definition 7. *Input Transitions and Output Transitions*
Let $\mathcal{A} = (Q, E, T, L, \ell, \mathrm{p})$ *be a SAN automaton.*
The functions inputs : $Q \to 2^T$ *and* outputs : $Q \to 2^T$ *that give the* input transitions *and* output transitions*, respectively, of a state of* $\mathcal{A}$ *are defined as*

$$\forall q \in Q, \quad \text{inputs}(q) = \{(p, q, e) \mid (p, q, e) \in T\} \text{ and}$$
$$\forall q \in Q, \quad \text{outputs}(q) = \{(q, r, e) \mid (q, r, e) \in T\}.$$

Definition 8. *Source State and Absorbing State*
Let $\mathcal{A} = (Q, E, T, L, \ell, p)$ *be a SAN automaton and* $q \in Q$ *a state of* $\mathcal{A}$*.*
If $\text{inputs}(q) = \emptyset$*, then* q *is a* source state*.*
If $\text{outputs}(q) = \emptyset$*, then* q *is an* absorbing state*.*

In this work, we propose an algorithm to automatically convert a well-defined BPMN Process model (Definition 4) to a SAN model in the format of Definition 6. This conversion starts with the individual mapping of the objects of the BPMN graph into SAN objects, according with the illustrations in Table 2. In a general way, we can say that the mapping of each vertex of the BPMN graph originates in the SAN model at least one new automaton, with at least two states and a transition between them (associated with a new event labeled with the identifier of the BPMN vertex). Event, activity, and exclusive gateway vertices generate only local events. Parallel gateways originates only synchronizing events. Inclusive gateways (that must appears in pairs, delimiting closed blocks) generate both local and synchronizing events.

Parallel sequence flows are mapped to sets of synchronized automata. A choice is expressed by a state that has more than one output transition (remembering that, in this case, each output transition must be weighted by a probability value). An atomic activity a is mapped into a 3-state sequential automaton, where the first state indicates that the activity is waiting for the availability of its required resources, the second state indicates that it had already obtained the access to the resources (by event r_a), and the third state indicates that the execution of the activity is finished (with the occurrence of event a).

In order to obtain the SAN model correspondent to a well-defined BPMN Process model, we must compound the simple automata generated from the mappings of Table 2. This composition is made by means of a set of *simplification operations* applicable over SAN models, that we define in the following.

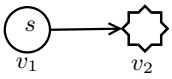
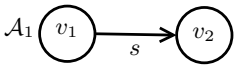
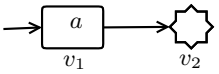
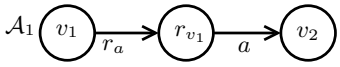
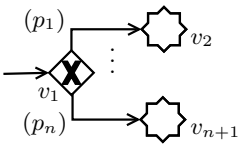
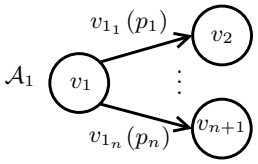
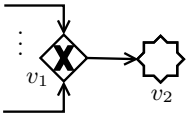
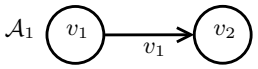
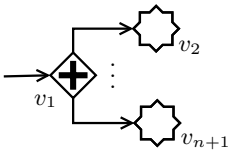
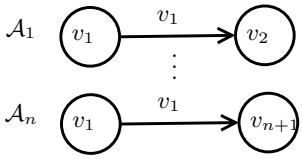
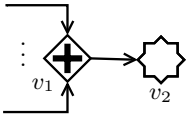
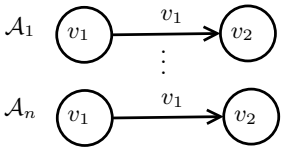
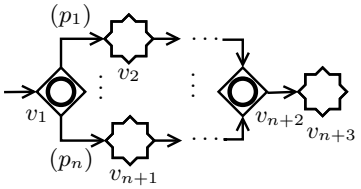
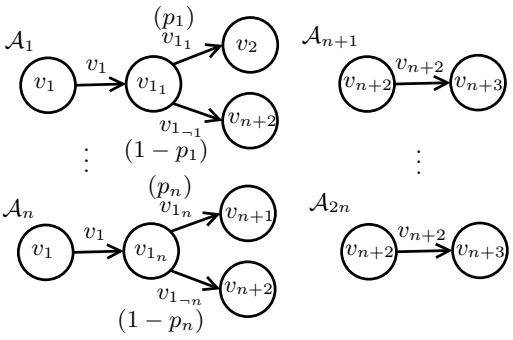
BPMN Object[1,2]	**SAN Mapping**
Start event labeled s 	
Atomic activity labeled a 	
Exclusive gateway diverging 1 sequence flow in n 	
Exclusive gateway converging n sequence flows in 1 	
Parallel gateway diverging 1 sequence flow in n 	
Parallel gateway converging n sequence flows in 1 	
Inclusive gateway block diverging 1 sequence flow in n and reconverging them in 1 again 	

Table 2. Mapping of the BPMN objects into SAN.

[1] We are using the symbol ⬡ to express any valid vertex of a well-defined BPMN model (i.e., an *atomic activity*, or an *end event*, or a *gateway*).

[2] It is important to notice that in this graphical representation of the BPMN graphs, we included two textual annotations in the vertices: a label (appearing inside the vertex), and an identifier v_i (appearing at the bottom of the vertex).

Definition 9. *State Merging Operation* ($\triangleright$)

Let $\mathcal{A} = (Q, E, T, L, \ell, \mathrm{p})$ be a SAN automaton and $q_1, q_2 \in Q$ be two states of $\mathcal{A}$.

The state merging *of q_1 and q_2 in $\mathcal{A}$ (represented by $\mathcal{A}[q_1 \triangleright q_2]$) results in a SAN automaton $\mathcal{A}_M = (Q_M, E_M, T_M, L_M, \ell_M, \mathrm{p}_M)$ such that:*

- $Q_M = Q \setminus \{q_2\}$
- $E_M = E$
- $T_M = \{(p, q, e) \in T \mid (p \neq q_2) \wedge (q \neq q_2)\} \cup \{(p, q_1, e) \mid (p, q_2, e) \in T\} \cup \{(q_1, q, e) \mid (q_2, q, e) \in T\}$
- $L_M = L$
- $\forall q \in Q_M, \ell_M(q) = \ell(q)$
- $\forall (p, q, e) \in T_M,\ \mathrm{p}_M((p, q, e)) = \begin{cases} \mathrm{p}((p, q, e)), & \text{if } (p, q, e) \in T \\ \mathrm{p}((p, q_2, e)), & \text{if } (q = q_1) \wedge ((p, q_2, e) \in T) \\ \mathrm{p}((q_2, q, e)), & \text{if } (p = q_1) \wedge ((q_2, q, e) \in T) \end{cases}$

This operation eliminates q_2 from $\mathcal{A}$, transforming all the input/output transitions of q_2 in input/output transitions of q_1 (keeping the label of q_1 unchanged).

Definition 10. *State Suppression Operation* ($\blacktriangleright$)

Let $\mathcal{A} = (Q, E, T, L, \ell, \mathrm{p})$ be a SAN automaton and $q \in Q$ be a state of $\mathcal{A}$ such that $|\mathrm{outputs}(q)| = 1$. Let t_o be the output transition of q and o be its output state.

The state suppression *of q in $\mathcal{A}$ (represented by $\mathcal{A}[q \blacktriangleright]$) results in a SAN automaton $\mathcal{A}_S = (Q_S, E_S, T_S, L_S, \ell_S, \mathrm{p}_S)$ such that:*

- $Q_S = Q \setminus \{q\}$
- $E_S = \{e \in E \mid (p, r, e) \in (T \setminus \{t_o\})\}$
- $T_S = \{(p, r, e) \in T \mid (p \neq q) \wedge (r \neq q)\} \cup \{(p, o, e) \mid (p, q, e) \in T\}$;
- $L_S = \{l \in L \mid \exists p \in Q, ((\ell(p) = l) \wedge (p \neq q))\}$
- $\forall q \in Q_S, \ell_S(q) = \ell(q)$
- $\forall (p, r, e) \in T_S,\ \mathrm{p}_S((p, r, e)) = \begin{cases} \mathrm{p}((p, r, e)), & \text{if } (p, r, e) \in T \\ \mathrm{p}((p, q, e)), & \text{in the other cases} \end{cases}$

This operation eliminates q and its output transition from $\mathcal{A}$, transforming all the input transitions of q in input transitions of its only output state o.

Definition 11. *Automata Concatenation Operation* ($\boxplus$)

Let $\mathcal{A}_1 = (Q_1, E_1, T_1, L_1, \ell_1, \mathrm{p}_1)$ and $\mathcal{A}_2 = (Q_2, E_2, T_2, L_2, \ell_2, \mathrm{p}_2)$ be two SAN automata. Let $q_1 \in Q_1$ be an absorbing state and $q_2 \in Q_2$ be a source state.

The concatenation of $\mathcal{A}_1$ and $\mathcal{A}_2$ via the states q_1 and q_2 *(represented by $\mathcal{A}_1 \underset{q_2}{\overset{q_1}{\boxplus}} \mathcal{A}_2$) is the SAN automaton $\mathcal{A}_C = (Q_C, E_C, T_C, L_C, \ell_C, p_C)$, where:*

- $Q_C = (Q_1 \cup Q_2) \setminus \{q_2\}$
- $E_C = E_1 \cup E_2$
- $T_C = T_1 \cup \{(q_1, p, e) \mid (q_2, p, e) \in T_2\} \cup (T_2 \setminus \{(q_2, p, e) \mid (q_2, p, e) \in T_2\})$

- $L_C = L_1 \cup L_2$
- $\forall q \in Q_C, \ell_C(q) = \begin{cases} \ell_1(q), & \textit{if} \quad q \in Q_1 \\ \ell_2(q), & \textit{in the other cases} \end{cases}$
- $\forall t = (p, q, e) \in T_C$, $\mathrm{p}_C(t) = \begin{cases} \mathrm{p}_1(t), & \textit{if} \quad t \in T_1 \\ \mathrm{p}_2((q2, q, e)), & \textit{if} \quad (p = q1) \wedge ((q_2, q, e) \in T_2) \\ \mathrm{p}_2(t), & \textit{in the other cases} \end{cases}$

3.3 The Conversion Algorithm

Algorithm 1 shows the main steps involved in the conversion of a well-defined BPMN Process model to a SAN model. It first creates a SAN model composed of all automata generated from the individual conversion of the vertices of the input BPMN graph. This individual conversion, made by function "ConvertVertexInAutomata"[4], is the implementation of the mappings described in Table 2. In the sequence, this SAN model is reduced using Procedure 1, which applies the operations defined in Section 3.2 to create the final SAN model.

The final number of automata in a SAN model generated by our conversion method from a well-defined BPMN model BP is given by Formula 2.

$$\begin{aligned} \text{numberOfAutomata}(BP) &= |S_{BP}| + \sum_{g \in G'} (|\,\text{outputs}(g)| - 1)\ , \\ \text{where } G' &= \{g \in G_{BP} \mid (|\,\text{outputs}(g)| > 1) \wedge (\ell_{BP}(g) \in \{\text{“}\times\text{”}, \text{“}\bigcirc\text{”}\})\} \end{aligned} \tag{2}$$

We can see each automaton as an independent sequence flow of the business process. For that, we have at least as many automata as the number of start events in BP ($|S_{BP}|$). In addition, for each divergent parallel or inclusive gateway, a new set of automata is required. The size of this set is given by the number of branches (outputs) of the divergent gateway less 1 (because one of the branches is treated as the continuation of the automaton that originates the divergence).

Algorithm 1 ConvertBPtoSAN(BP)

Input: BP – a well-formed BPMN graph that is also a well-defined Process model
Output: $\mathcal{S}$ – a SAN model

1: $\mathcal{S} \leftarrow \emptyset$
2: $\mathcal{V} \leftarrow S_{BP} \cup A_{BP} \cup G_{BP} \cup F_{BP}$ {All vertices of the BP graph}
3: **for all** $v \in V$ **do**
4: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup$ ConvertVertexInAutomata(BP, vertex, $\mathcal{S}$)
5: **end for**
6: ReduceSANModel(BP, $\mathcal{S}$)
7: **return** $\mathcal{S}$

[4] We will not detail this function in an algorithmic form because of space restrictions.

Procedure 1 ReduceSANModel(BP, $\mathcal{S}$)

Input: BP – a well-formed BPMN graph
Input/Output: $\mathcal{S}$ – a SAN model

1: { Concatenate the "sequential" automata}
2: **while** there is an absorbing state $q_1 \in \mathcal{A}_1$ and a source state $q_2 \in \mathcal{A}_2$ (with $\mathcal{A}_1, \mathcal{A}_2 \in \mathcal{S}$) such that $\ell_{\mathcal{A}_1}(q_1) = \ell_{\mathcal{A}_2}(q_2)$ **do**
3: $\quad \mathcal{A}_C \leftarrow \mathcal{A}_1 \overset{q_1}{\underset{q_2}{\boxplus}} \mathcal{A}_2$ {Concatenate the two automata}
4: $\quad$ { Merge the equivalent states correspondent to the converging exclusive gateways}
5: $\quad$ **while** $\exists\ q_1, q_2 \in Q_{\mathcal{A}_C}$ such that $\ell_{\mathcal{A}_C}(q_1) = \ell_{\mathcal{A}_C}(q_2)$ **do**
6: $\quad\quad \mathcal{A}_C \leftarrow \mathcal{A}_C[q_1 \rhd q_2]$
7: $\quad$ **end while**
8: $\quad \mathcal{S} \leftarrow (\mathcal{S} \setminus \{\mathcal{A}_1, \mathcal{A}_2\}) \cup \mathcal{A}_C$
9: **end while**
10: **for all** $\mathcal{A} \in \mathcal{S}$ **do**
11: $\quad$ { Remove the states created to link alternative sequence flows }
12: $\quad$ **while** $\exists\ q \in Q_{\mathcal{A}}$ and $\exists v \in G_{BP}$ such that: $(\ell_{\mathcal{A}}(q) = v \ \wedge\ \ell_{BP}(v) = \text{“}\times\text{”} \ \wedge\ |\,\text{outputs}(v)| = 1)$ **do**
13: $\quad\quad \mathcal{A} \leftarrow \mathcal{A}[q \blacktriangleright]$ { Suppress state q}
14: $\quad$ **end while**
15: $\quad$ { Merge the source state with the absorbing states}
16: $\quad$ **while** $\exists q_2 \in Q_{\mathcal{A}}$ such that q_2 is an absorbing state **do**
17: $\quad\quad q_1 \leftarrow$ the only source state of $\mathcal{A}$
18: $\quad\quad \mathcal{A} \leftarrow \mathcal{A}[q_1 \rhd q_2]$ { Merge the states q_1 and q_2}
19: $\quad$ **end while**
20: **end for**

Two remarks must be made about our conversion algorithm. The first one is that the SAN models generated by our algorithm are not the only ones possible to represent the BPMN models provided as input. The second one is that the order in which the pair of automata are selected to be concatenated (lines 2 and 3 of Procedure 1) impacts the final SAN models. Different orders may generated structurally different SAN models that are equivalent in terms of modeled behavior. These different models can also imply in different difficulty levels to numerically solve the model.

The SAN model generated by our conversion algorithm reflects the behavior of one instance of the business process, disregarding the resource usage. To analyze the behavior of the system when several instances are being executed in parallel, we need to replicate the automata of the SAN model – each replica represents an instance of the process; SAN already counts on the concept of replicas [3].

Implementation The conversion method was implemented as part of a software tool that is available at `http://www.ime.usp.br/~kellyrb/bp2san`. This tool receives as input a BPMN model (textually described using the DOT language [10]) and generates behaviorally equivalent SAN models corresponding to

the given business process. The resulted SAN models are textually expressed using the syntax accepted by the PEPS tool [5].

The automatically generated SAN models can be further annotated, for example, with the information about the rates associated to the events or with additional automata expressing resource constraints. This will allow a complete numerical analysis of the model through the PEPS tool.

As one can imagine, to know the appropriate rate/probability to be associated to each event of the stochastic model of a business process may not be a trivial task. In some cases, the rates/probabilities can be defined by a business specialist, based in his/her knowledge of the domain. In other cases, when the real system is already implemented and in use, these values can be inspired by the times/frequencies observed in the real system. However, it is important to emphasize that one of the great advantages of the performance analysis via analytical modeling is the *predictive* role it can assume. In this kind of analysis technique, it is easy to change the parameters of the model to evaluate how the system will react to different workloads or different resource availabilities. For this reason, even when the exact rates/probabilities for the events are not known at the modeling time, multiple analyses (made over the same model, but with different parameters) may help in the tunning of the system, because they can indicate the characteristics required for the system to meet the expected performance.

3.4 Example – A Simple Model with an Inclusive Choice

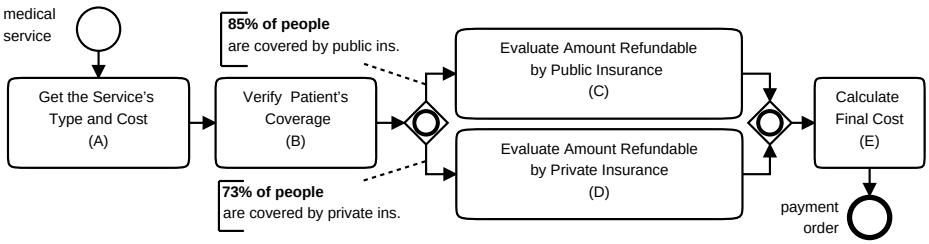


Fig. 1. BPMN model of the french process to determine the cost of a medical service.

Fig. 1 shows a simple model in BPMN that recalls an example given in [4]. It is a process existing in the french health-care system, to define the final cost of a medical service to a patient. The health-care system in France involves a mix of public and private financing. The public financing offers the coverage of basic medical services. But the French may also buy supplemental insurance which reduces their out-of-pocket costs and possibly covers extra expenses. The SAN model generated from the conversion of the vertices in Fig. 1 is shown in Fig. 2.

After the conversion of the vertices, the generated SAN model is reduced according the steps defined in the Procedure 1. Fig. 3 shows one of the possible

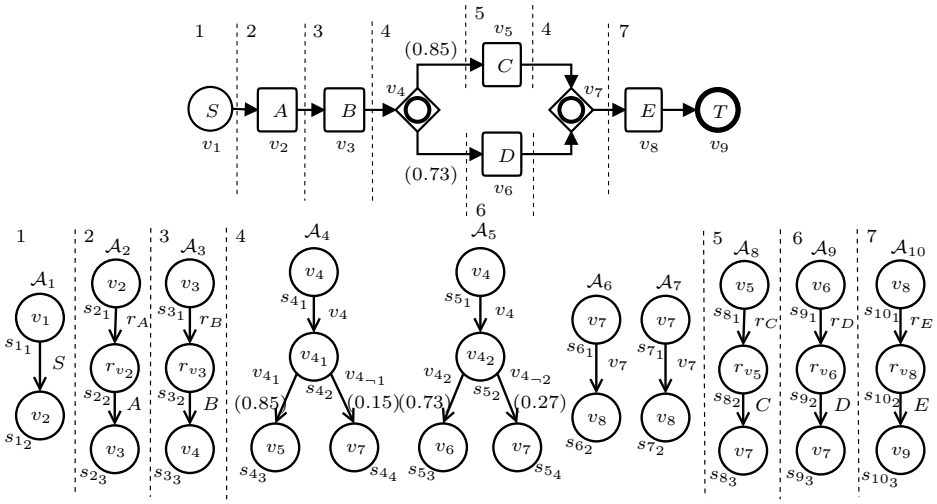


Fig. 2. SAN model obtained after the conversion of the BPMN vertices.

SAN models resulted from the reduction. This model can be originated by the two following sequences of operations:

$$\mathcal{A}' = ((\mathcal{A}_1 \underset{s_{2_1}}{\overset{s_{1_2}}{\boxplus}} \mathcal{A}_2 \underset{s_{3_1}}{\overset{s_{2_3}}{\boxplus}} \mathcal{A}_3 \underset{s_{4_1}}{\overset{s_{3_3}}{\boxplus}} \mathcal{A}_4 \underset{s_{6_1}}{\overset{s_{4_4}}{\boxplus}} \mathcal{A}_6 \underset{s_{8_1}}{\overset{s_{4_3}}{\boxplus}} \mathcal{A}_8)[s_{4_4} \rhd s_{8_3}] \underset{s_{10_1}}{\overset{s_{6_2}}{\boxplus}} \mathcal{A}_{10})[s_{1_1} \rhd s_{10_3}]$$

$$\mathcal{A}'' = ((\mathcal{A}_5 \underset{s_{7_1}}{\overset{s_{5_4}}{\boxplus}} \mathcal{A}_7 \underset{s_{9_1}}{\overset{s_{5_3}}{\boxplus}} \mathcal{A}_9)[s_{5_4} \rhd s_{9_3}])[s_{5_1} \rhd s_{7_2}]$$

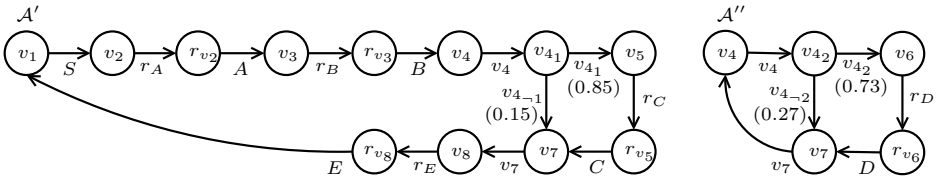


Fig. 3. One of the SAN models that can be obtained from the reduction of Fig. 2.

Even in this simple example is possible to notice the reduction of memory usage provided by SAN models. The model in Fig. 3 has an underlying Markov chain with 24 states (the reachable states resulted from the cartesian product of the sets of states of the two automata). The infinitesimal generator of this Markov chain is a 24×24 matrix (576 cells). However, with SAN this matrix is not stored – only the infinitesimal generator of each automaton is stored (in the case of the example, a 12×12 and a 5×5 matrices, totaling 169 cells).

4 Related Works

Several works proposed mappings from business process models to formal languages [1, 8, 9, 18]. However, most part of these works concern non-stochastic formalisms. In this section, we will restrict ourselves to the works related to the stochastic modeling of business processes.

Reijers [17] presented two interesting analytical methods for performance evaluation of *Workflow Nets* – a sub class of Petri Nets specially defined to workflow modeling. His methods deal with resource management, but under the assumption of an infinite amount of resources.

The work of Canevet et al. [7] proposed an automated mapping from the *Unified Modeling Language* (UML) [13] state diagrams enhanced with performance information to *Performance Evaluation Process Algebra* (PEPA) [11]. The performance information they refer are probabilities attached to the states and rates attached to the transitions of the UML state model. One important advantage of the approach proposed by the authors is that the performance results obtained from the solution of the PEPA model can be reflected back to the UML level. However, the approach does not support functional rates, preventing some important aspects related to performance from being contemplate in the modeling.

The proposal of Prandi et al. [16] was a mapping from BPMN to *Calculus for Orchestration of Web Services* (COWS), a process calculus inspired by the *Business Process Execution Language* (BPEL). The authors made a briefly discussion about the use of a stochastic extension of COWS to support quantitative analysis of business processes. Despite being based in a compositional formalism, Stochastic COWS does not explore the advantage of the compositionality in the analysis method, thus suffering of the same state-space explosion problem that limits the use of other Markovian formalisms in the analysis of large-scale systems.

Braghetto et al. [4] discussed the viability of applying three different stochastic formalisms – the *Generalized Stochastic Petri Nets* (GSPN) [2], PEPA and SAN –, in the modeling for numerical analysis of performance of business processes initially modeled using BPMN. They verified that the three studied stochastic formalisms are able to express with equivalent facilities basic business process scenarios modeled in BPMN. However, more advanced scenarios evidenced the pros et cons of each formalism. Since SAN and PEPA are intrinsically compositional formalisms, they enable a structured analysis, in addition to the facility to extend a model without impacting the previous modeled behavior. SAN and GSPN have the explicit notion of state and the concept of functional rates, what helps to model functional dependencies between the components of a process.

The study made in [4] was the first one considering the use of SAN to model business processes. The SAN efficiency and expressiveness motivated us to extend these first results, with the proposal of an automated method to map from BPMN to SAN.

5 Concluding Remarks

In this work, we proposed an algorithm to automatically convert a subclass of the BPMN Process diagrams into SAN models. For this, we formally defined the characteristics of a well-defined BPMN Process model, mappings of BPMN objects into elementary SAN models, and a set of operations over SAN models (to transform the elementary models in the final SAN model of the business process). This conversion is the first step to build a complete performance evaluation model of business processes. The second step is to enrich the resulted SAN model with resource management information, to obtain the correct rates associated with the events that represent the activities of the business process.

As seen in Section 2.1, SAN automata are constituted by states and transitions between these states. Transitions are triggered by events; each event has an associated rate (that indicates its frequency of occurrence in function of the time). However, in the structure of the SAN models that we handled in Section 3, we did not include rates. The definition of the correct rates for events generated from a business process is intrinsically related to the resource management of the process and it is a subject for further deeper studies.

Our ongoing work consists in: (i) the definition of annotations over BPMN models to specify the resource requirements of each activity and how these resources are shared between activities executed parallelly; and (ii) an automated method to complement the SAN model of the business process with the information of these annotations. With this, we will be able to automatically obtain SAN models from business process models that will be able to provide performance indicators that really approximate the results expected for the business process in the real world.

References

1. van der Aalst, W.M.P.: Formalization and verification of event-driven process chains. Information and Software Technology 41(10), 639 – 650 (1999)
2. Balbo, G.: Introduction to generalized stochastic Petri nets. In: SFM 2007: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems. LNCS, vol. 4486, pp. 83–131. Springer (2007)
3. Benoit, A., Brenner, L., Fernandes, P., Plateau, B.: Aggregation of stochastic automata networks with replicas. Linear Algebra and its Applications 386, 111 – 136 (2004)
4. Braghetto, K.R., Ferreira, J.a.E., Vincent, J.M.: Performance analysis modeling applied to business processes. In: Proceedings of the 2010 Spring Simulation Multiconference. pp. 122:1–122:8. SpringSim '10, ACM (2010)
5. Brenner, L., Fernandes, P., Plateau, B., Sbeity, I.: PEPS2007 – stochastic automata networks software tool. In: Fourth International Conference on the Quantitative Evaluation of Systems. pp. 163 –164. QEST 2007 (2007)
6. Brenner, L., Fernandes, P., Sales, A.: The need for and the advantages of generalized tensor algebra for Kronecker structured representations. International Journal of Simulation: Systems, Science & Technology 6(3-4), 52–60 (2005)

7. Canevet, C., Gilmore, S., Hillston, J., Prowse, M., Stevens, P.: Performance modelling with the unified modelling language and stochastic process algebras. Computers and Digital Techniques, IEE Proceedings - 150(2), 107 – 120 (Mar 2003)
8. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology 50(12), 1281 – 1294 (2008)
9. Eshuis, H.: Semantics and Verification of UML Activity Diagrams for Workflow Modelling. Ph.D. thesis, Univ. of Twente (November 2002)
10. Gansner, E.R., North, S.C.: An open graph visualization system and its applications to software engineering. Softw. Pract. Exper. 30, 1203–1233 (September 2000)
11. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, New York, NY, USA (1996)
12. OMG: Business process model and notation (BPMN), version 2.0 (2010)
13. OMG: Unified modeling language specification (UML), version 2.3 (2010)
14. Plateau, B.: On the stochastic structure of parallelism and synchronization models for distributed algorithms. SIGMETRICS PER 13(2), 147–154 (1985)
15. Plateau, B., Atif, K.: Stochastic automata network for modeling parallel systems. Software Engineering, IEEE Transactions on 17(10), 1093 –1108 (1991)
16. Prandi, D., Quaglia, P., Zannone, N.: Formal analysis of BPMN via a translation into COWS. In: Proceedings of COORDINATION'08. pp. 249–263. Springer (2008)
17. Reijers, H.A.: Design and control of workflow processes: business process management for the service industry. Springer-Verlag, Berlin, Heidelberg (2003)
18. Wong, P.Y., Gibbons, J.: A process semantics for BPMN. In: Proceedings of the 10th International Conference on Formal Methods and Software Engineering. pp. 355–374. ICFEM '08, Springer-Verlag (2008)