Graph-based analysis of business process models

PIET BOEKHOUDT, HENK JONKERS, MICHIEL ROUGOOR
Telematica Instituut
P.O. Box 589
7500 AN Enschede
The Netherlands

Abstract: - It is widely accepted that using models is effective for the analysis of business processes. Analysis of business process models is useful for clarifying business process characteristics, for identifying bottlenecks, and for comparison of alternatives. Among the modelling languages, those who have a formal semantic basis are the most suitable for analysis. The AMBER modelling language, which is explained in this paper, has such a formal basis. By way of an example we show that the AMBER modelling language has sufficient expressive power for business process modelling. Furthermore, we show that a class of AMBER models can easily be transformed to a graph representation. This graph representation is a starting point for graph-based analysis, which is generic in the sense that it is independent of AMBER specific concepts and suitable for different types of analysis. In this paper we present a graph reduction algorithm which can be used for computing overall business process characteristics (such as the completion time distribution) and path characteristics (such as probabilities of all possible process instances).

Key-Words: - business processes, models, business process (re)engineering, analysis, graphs

1. Introduction

External developments force companies to continuously improve their business processes. Change of running processes is hazardous and might even endanger the continuity of the company. Therefore, an offline approach to business process improvement is necessary to evaluate different process design alternatives. The use of models is a widely accepted way of doing offline (re)design and analysis of business processes. Models help to better understand and communicate process descriptions. Furthermore, analysis of models is possible, provided that the modelling language has a formal mathematical basis.

In this paper we present research results from the Testbed project [1]. In the Testbed project a number of functional and quantitative model analysis methods for so-called AMBER models have been developed (cf. [2], [3], and [4]). Examples of these analysis methods are critical path analysis, completion time analysis, and utilisation and queueing analysis. The AMBER modelling language and the analysis methods have been implemented in a professional business process engineering tool, which is used by the Testbed consortium partners. The model-based approach, as presented in this paper, has proved invaluable in many practical cases [1].

In this paper we present a graph-based formalism which can be used for different types of business process analysis. We present a graph reduction algorithm which can be used for computing overall business process characteristics (such as the completion time distribution) and path characteristics (such as probabilities of all possible process instances).

The organisation of this paper is as follows. In Section 2 the AMBER modelling language is explained. In Section 3 we describe how an AMBER model can be translated to a graph representation. In Section 4 we present a generic graph based algorithm for analysis. In Section 5 we present our conclusions and our future work. Throughout this paper a simple business process model will be used for illustration.

2. The AMBER Modelling Language

In the Testbed approach [1], we distinguish three modelling domains, corresponding to different aspects of business processes. The *behaviour* domain models the activities and their relationships, the *entity* domain (or: *actor* domain, as used in later publications) models the resources that carry out these activities, and the *item* domain models the data objects which are manipulated by the process. The latter two domains are not relevant for the types of analysis described in this paper, and will therefore not be considered. The modelling language in Testbed is based on an architectural design framework for distributed systems developed at the University of Twente [6] and is called AMBER (Architectural Modelling Box for Enterprise Redesign). In the remainder of this section, we will introduce the main concepts in the behaviour domain, and illustrate them on an example process.

2.1 Actions and causality relations

The basic behaviour element is an action, denoted by a (stretched) circle. Actions can be related in that certain restrictions can be imposed on the order in which they are performed. Fig. 1 shows the basic relations between actions that can be expressed in the language.

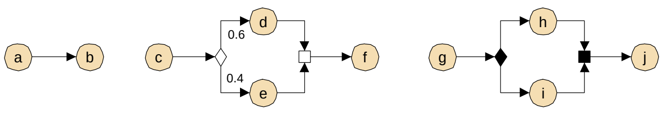

Fig. 1 Basic relations: causality, choice, parallelism.

The simple *causality* relation indicates that action *b* can only occur after action *a* has completed. *Choice* is modelled by an *or*-split (open diamond): after action *c* has completed, either *d* or *e* can occur. A condition or probability can be associated with the branches. The *or*-join (open box) indicates that action *f* can start if either *d* or *e* has finished. *Parallelism* is modelled by an *and*-split (black diamond): after action *g* has completed, both *h* and *i* can start. The *and*-join (black box) indicates that *j* can start as soon as both *h* and *i* have finished.
The splits and joins can be used in any combination: it is, e.g., not required that a section that starts with an *and*-split is followed by an *and*-join. However, the majority of models found in practice consist of a structured (possibly nested) use of matching splits and joins. We call these models basic *series-parallel* (SP) models, because they form an extension of the well known class of SP task graphs as described in, e.g. [7]. The analysis techniques described in this paper apply for basic SP models, as well as for models that can be transformed to these models (extended SP models, to be explained in section 3.2).

2.2 Structured models and interactions

In addition to the above-mentioned basic ingredients of a behaviour model, Testbed offers a number of concepts to build models in a more structured way. In particular, a model can be subdivided in a number of *behaviour blocks*, drawn as rounded rectangles. A behaviour block groups a number of actions, and can contain other (nested) blocks. Actions can be grouped in many ways, e.g., based on the different resources (actors) that are involved in the process. *Interactions*, represented by (stretched) semicircles, are actions that are performed by two (or more) co-operating behaviour blocks. Corresponding interaction contributions are connected.
Fig. 2 shows the behaviour model of an example that will be used throughout this paper. The numbers in this figure are probabilities. In this example the process of ordering and receiving office goods is modelled:

The process starts with a need for goods of office employees, who send a request to the office secretary (*request*). The secretary sends an order to a supplier (*send*). The supplier ships the goods and delivers the goods to the goods department of the office where they are received (*receive*). At the goods department the goods are inspected for completeness and possible damage (*inspect*). If goods are OK, they are booked, forwarded and finally stored by the office employees (*store*). If however, after inspection, the goods turn out to be not OK a compensation order is made by the secretary (*renew*). The secretary gets completion details from office employees (*detail*). The complete compensation order is send to the supplier (*resend*). Simultaneously, an investigation is made whether the goods can be reutilized (*list*). If goods *can not* be reutilized, they are returned to the supplier (*return*). If goods *can* be reutilized, they are booked and forwarded to the department where the employees store them (*store*).

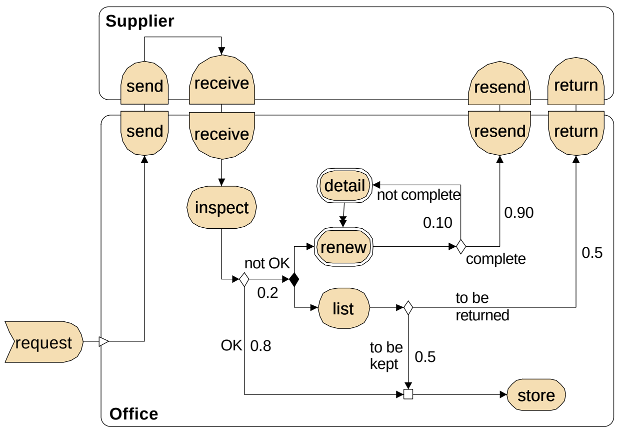

Fig. 2 Example: goods ordering process.

This model shows a number of concepts that have not yet been explained:

- A *trigger* starts a process, in this example *request* (office employees having a need for certain goods).
- *Entries* and *exits*, represented by small triangles, are used when an arrow crosses the boundaries of a behaviour block
- *Repetitive* (inter)actions, i.e. (inter) actions that (potentially) occur more than once are drawn with a double edge. A double-pointed arrow indicates the start of a new loop. In the example the compensation order is defined in an iterative fashion: details are added until the compensation order is evaluated complete.

The purpose of the subdivision of the model in behaviour blocks is to increase the readability of the model. It is irrelevant for the types of analysis described in this paper: interactions connected with an interaction relation can be considered as a single action, and an incoming and outgoing arrow of an entry or exit can be combined into a single arrow.

2.3 Profiles

The concepts as described above only describe the existence of actions and their causal relations. However, for the analyses we will consider next, we also need to know their properties, such as the duration of actions and branching probabilities for an *or*-split. In the AMBER language, properties specific for a certain type of analysis are attached to the concepts by so-called *profiles*. For completion time analysis, for example, we attach a “Completion Time”-profile to each (inter)action, consisting of a number of attributes, such as the mean duration and the probability distribution type. For causality arrows leaving an *or*-split we attach probabilities in a “Probability”-profile.

3. Graph representation of models

In this paper, we consider analysis methods that use the *structure* of a process model in terms of (nested) serial, parallel (“and”), choice (“or”) and loop (“repetition”) constructs. However, the models in the previous section do not directly show this structure. Therefore, we propose an intermediary representation of models as a directed, acyclic *series-parallel* (SP) graph, where the nodes represent the above-mentioned constructs and the leaves represent actions. Depending on the type of analysis, it might be needed to annotate a node with certain additional information, e.g. the duration of an action. For many analysis techniques the outgoing arcs of an *or*-node should be annotated with a probability. Note that models can only be represented in this way if it is possible to describe them as a (nested) combination of these constructs. Therefore, the analysis techniques that we will describe are only applicable to this class of models, which we will call SP models.

3.1 Tree representation of basic SP models

First, we will consider the class of models that we will call *basic* SP models, i.e., models for which the graph representation is a *tree* rather than a general directed acyclic graph.

Fig. 3 shows the basic constructs and their equivalent in an SP tree.

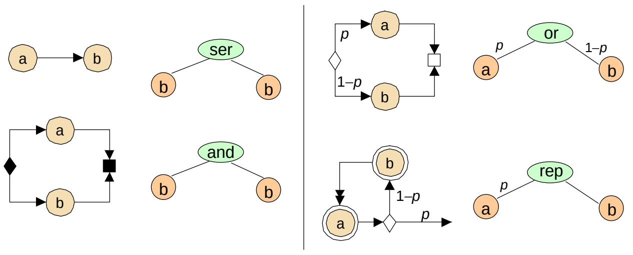

Fig. 3 SP tree representations of basic constructs.

3.2 Graph representation of extended and approximate SP models

We can extend the class of SP models by considering models that can be replaced by an equivalent basic SP model by means of duplication of actions, i.e., an action can occur in more than one parallel or choice section. In the SP-graph, this means that a leaf or intermediary node can have more than one incoming arc, i.e., the graph is no longer a tree. Fig. 4 shows a typical example of such a model and its SP tree representation, as well as the equivalent basic SP model (where the double line indicates that the actions *c* are identical, i.e., they correspond to the same action in the original model).

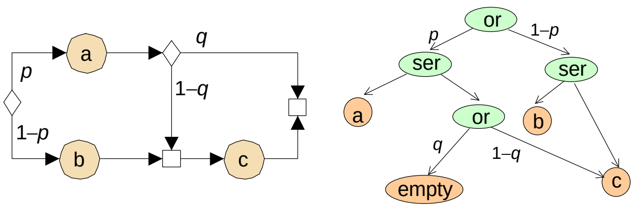

Fig. 4 SP-graph representation of an extended SP-model

Finally, we consider models that we will call *approximate* SP models. For these models, it is not possible to construct an equivalent basic SP model. However, by means of action duplication, we can construct a basic SP model that can be used to obtain exact or approximate results for some types of analysis (e.g., exact results for critical path analysis, and approximate results for stochastic completion time analysis). Fig. 5 shows a typical example of an approximate SP-model and its SP graph representation, as well as the basic SP model that approximates the original model.

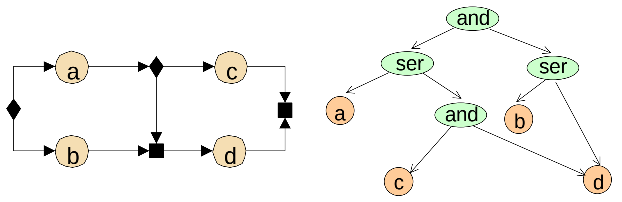

Fig. 5 SP graph representation of an approximate SP model.

3.3 Derivation of an SP graph

In this subsection we illustrate how an SP graph is constructed, given an AMBER model of a process. We start with a "flat" model, i.e., a model in which the block structure has been removed and related interactions have been combined into single actions. Fig. 6 shows the flat version of our example model.

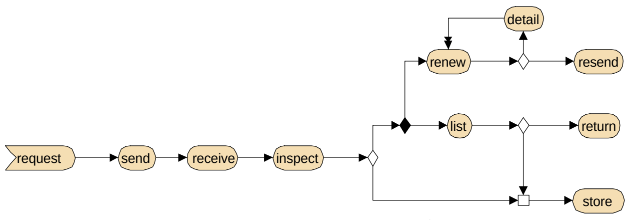

Fig. 6 Original model ("flat").

The general idea of the algorithm is to reduce the AMBER model, and at the same time building up the SP graph. In the original model, we identify *sections*, i.e., arrows between *nodes* (where nodes can be actions, splits or joins). With each section, we can associate an SP graph (which is a part of the SP graph of the whole model). In the initial model, these graphs are empty.
The AMBER model is reduced by combining several sections into one new section, while storing the structural information in the SP graph associated with the new section. Four types of reduction are defined: *serial*, *split* (which can be refined into *parallel* and *choice*), *repeat* and *duplication*. Fig. 7 shows an example of the result of a number of serial reductions.

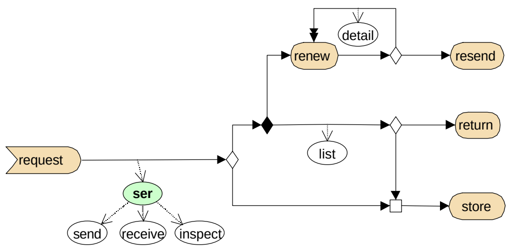

Fig. 7 Partly reduced graph.

Duplication occurs in extended SP models, e.g. for the action *forward* in Fig. 8. The result of duplication is that one SP graph is associated with more than one section (in our example this is the graph consisting of the single node *forward*). Fig. 8 also shows examples of the result of a *repeat* reduction.

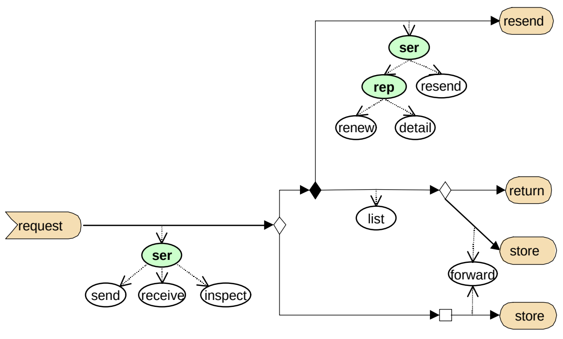

Fig. 8 Further reduced graph.

The reduction continues until a single section remains. The SP graph associated with that section is the SP graph that represents the complete model. Sometimes it is not possible to reduce an AMBER model to a single section: this means that the model is not a (basic or extended) SP model and the analysis methods described in this paper cannot be applied to this model. Fig. 9 shows the final SP graph representation of our example model.

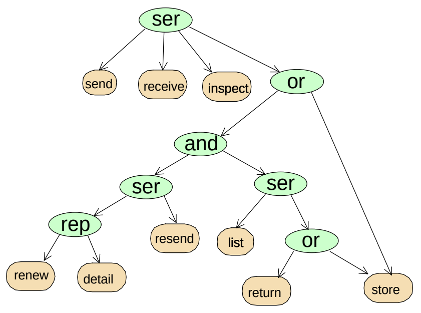

Fig. 9 SP graph representation of the example model.

4. Graph based analysis

In the previous section we transformed AMBER models to graphs. In this section we show how to perform different types of analysis on SP graphs. Basically, we traverse the tree and use a graph reduction algorithm. In the traversal step we navigate from the leaves to the root of the tree. In the reduction step we apply the analysis to elementary tree constructs.

4.1 Reduction algorithm

In this section we first consider the reduction of basic model constructs, like those in Fig. 3 The reduction amounts to substitution of the basic model construct by a single action and by computing the properties of this single action. An example should make things clear.

Example 1
Suppose we have the simple model as described in Fig. 10. The boxed numbers in this diagram are the processing times of the actions.

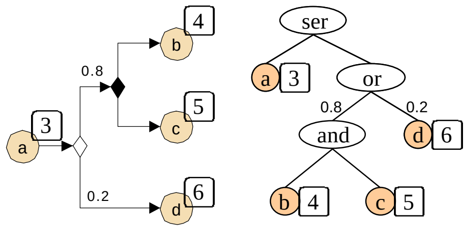

Fig. 10 Simple example and corresponding tree.

Suppose we wish to know the mean completion time of the process. We start by looking at the parallel section with actions *b* and *c*. As action *c* takes longer to complete than action *b* the duration of this section is max(4,5) = 5, assuming deterministic processing times. The reduction step now is to replace the parallel section by an action s_1 with processing time 5, as shown in Fig. 11.

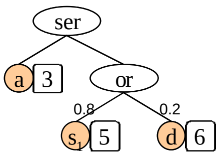

Fig. 11 First reduction step.

The second reduction step concerns the *or*-section with actions s_1 and *d*. Using the probabilities, we find for the mean completion time 0.8∗5 + 0.2∗6 = 5.2. The *or*-section is replaced by an action s_2 with processing time 5.2, as shown in Fig. 12.

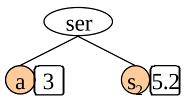

Fig. 12 Second reduction step.

In the final reduction step, the serial section is replaced by an action s_3. The processing time of s_3 equals the sum of the processing times of *a* and s_3, i.e. 8.2. In this final reduction step we find that the mean completion time of the process equals 8.2.

As this example shows, in a reduction step a section is replaced by a single action and a certain calculation on the attribute values is performed.
For certain types of analysis we also need to keep a record of the intermediate results. For example, if we wish to determine the critical path of the model in Fig. 10, we also need to exclude action *b* (which has a shorter duration than action *c*) as a result of the reduction. The actions on the critical path cannot be determined before the complete model is reduced to a single action. Actions that are not excluded are candidates for the critical path. For other types of analysis, we also wish to determine all possible paths and, simultaneously, for each individual path certain properties (such as the probability that an instance of this path will occur and the duration of this path).
It is for these reasons that we represent an action *a* (which might be a result of a reduction step) by the following quadruple:

$$(\text{a}, n, S_a, P_a),$$

where a is the name of the action, *n* is the number of alternative paths in the sub-model that is represented by *a*, S_a is the intermediate result for *a*, and P_a is a symbolic representation of the alternative paths including the properties of these paths. This is written as

$$P_a = <s_1, p_1, \ldots , s_i, p_i, \ldots, s_n, p_n>,$$

where s_i is the property of the *i*-th path p_i. The path of an *elementary* action *a* (i.e., an action which is not a result of an reduction step) is represented by a string

a;

where clearly, a is the name of the action and the semicolon is used as delimiter. The notation for the basic constructs is now given in Table 1.

Serial	$<s$, {a; b;}>
And	$<s$, [a; b;]>
Or	$<s_1$, a;, s_2, b;>
Rep	$<s_1$, a;, s_2, {a;*{b;a;}}>

Table 1 Notation for basic constructs.

If the probability of the occurrence of a path is the only property we are interested in, then in Table 1

$$s = 1,\ s_1 = p,\ s_2 = 1 - p.$$

The * in the notation for the repetition is used to express that the part following the * is executed an unspecified number of times.

We are now ready to give the *description* d_a of an elementary action a as

$$d_a = (\mathrm{a}, 1, S_a, <s_a, \mathrm{a};>).$$

The reduction of the basic model constructs now amounts to an operation on the descriptions of the actions, i.e. the reduction of a serial of elementary actions a and b is symbolically denoted by

ser_red(*a*,*b*)

Let

$d_a = (\mathrm{a}, 1, S_a, <s_a, \mathrm{a};>)$

$d_b = (\mathrm{b}, 1, S_b, <s_b, \mathrm{b};>)$.

Create a new action s, with

$d_s := (\mathrm{s}, 1, S_s, P_s)$,

and

$(S_s, S_a, S_b) := f_{\mathrm{serial}}(S_a, S_b)$,

$P_s := <g_{\mathrm{serial}}(s_a, s_b)$, {a;b;}>.

f_{serial} and g_{serial} denote operations, specific for the reduction of the serial, and specific for the type of analysis to be performed. Note that the reduction step may also alter the attribute values of a and b (for example in the critical path analysis, where an action may be excluded).

Clearly, the reduction of the basic constructs can symbolically be represented as in Table 2:

ser_red(a,b**)**	$d_s := (\mathrm{s}, 1, S_s, P_s)$ $(S_s, S_a, S_b) := f_{\mathrm{serial}}(S_a, S_b)$ $P_s := <g_{\mathrm{serial}}(s_a, s_b)$, {a;b;}>
and_red(a,b**)**	$d_s := (\mathrm{s}, 1, S_s, P_s)$ $(S_s, S_a, S_b) := f_{\mathrm{and}}(S_a, S_b)$ $P_s := <g_{\mathrm{and}}(s_a, s_b)$, [a;b;]>
or_red(a,b,p**)**	$d_s := (\mathrm{s}, 2, S_s, P_s)$ $(S_s, S_a, S_b) := f_{\mathrm{or}}(S_a, S_b)$ $P_s := <g_{\mathrm{or},1}(s_a, s_b, p)$, a;, $g_{\mathrm{or},2}(s_a, s_b, 1\text{-}p)$, b;>
rep_red(a,b,p**)**	$d_s := (\mathrm{s}, 2, S_s, P_s)$ $(S_s, S_a, S_b) := f_{\mathrm{rep}}(S_a, S_b)$ $P_s := <g_{\mathrm{rep},1}(s_a, s_b, p)$, a;, $g_{\mathrm{rep},2}(s_a, s_b, 1\text{-}p)$, {a;*{b;a;}}>

Table 2 Reduction of basic constructs.

Example 1 (continued)

Suppose we want to further analyse the simple model in Fig. 10, i.e. we want to determine the following:

- the actions on the critical path, i.e. the sequence of actions that dominate the completion time;
- the completion time of the critical path;
- all possible paths and the completion time of these paths;
- the probability of all possible paths.

For the description of an action s we take

$$d_s = (\mathrm{s}, n, S_s, <s_1, p_1, \dots, s_i, p_i, \dots, s_n, p_n>).$$

S_s has two components

- the completion time of the sub-model corresponding to action s (which might be a result of a reduction step);
- a Boolean variable which indicates whether s is to be excluded from the critical path.

s_i also has two components

- the completion time of the ith path;
- the probability of the occurrence of the ith path.

For the description of an elementary action a we therefore take

$$d_a = (\mathrm{a}, 1, S_a, <s_a, \mathrm{a};>).$$

where

$$S_a = (t_a, b_a),\ s_a = (d_a, p_a).$$

t_a is the completion time of a, and b_a is a Boolean variable which has value 0 if action a is excluded from the critical path. For an elementary action the completion time of the path (which is a single action) is $d_a = t_a$, and $p_a = 1$ (by default).

The first reduction step (cf. Fig. 13) deals with the elementary actions b and c, with descriptions

$$d_b = (\mathrm{b}, 1, (4, 1), <(4,1), \mathrm{b};>),$$

$$d_c = (\mathrm{c}, 1, (5, 1), <(5,1), \mathrm{c};>).$$

Action c has a larger completion time than action b, and therefore b is excluded. The result of the reduction is an action s_1 (cf. Fig. 11) with following description:

$$d_{s1} = (\mathrm{s}_1, 1, (5, 1), <(5, 1), [\mathrm{b};\ \mathrm{c};]>).$$

Since action b is excluded, also its description is changed:

$$d_b = (\mathrm{b}, 1, (4, 0), <(4,1), \mathrm{b};>),$$

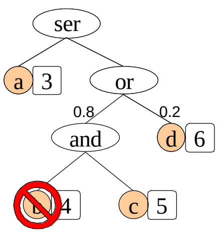

Fig. 13 First reduction step for critical path analysis.

The result of the reduction step is shown in Fig. 11. The second reduction step yields an action s_2 (cf. Fig. 12) representing two alternative paths:

d_{s2} = (s_2, 2, (5.2, 1), <(5, 0.8), [b; c;], (6, 0.2),d;>).

Finally, the reduction of the serial in Fig. 12 yields an action s_3 with description

d_{s3} = (s_3, 2, (8.2, 1), <(8, 0.8), {a;[b; c;]}, (9, 0.2),{a;d;}>).

From the description of s_3 we conclude that the mean processing time of the critical path is 8.2. There are two alternative paths, the first path has probability 0.8 and completion time 8; the second path has probability 0.2 and completion time 9. The critical path is constructed in the opposite direction, i.e. by deselecting all actions that constituted a deselected (substitute) actions. For this particular example this was already done for action *b*, which is an elementary action. Hence, all elementary actions except action *b* lie on the critical path.

If action *b* was not an elementary action, but for example a result of a reduction step for a parallel section with actions *e* and *f*, then these actions would be excluded as well.

The reduction algorithm consists of a generic part, which will be used for all types of analysis, and a specific part. The generic part deals with the construction of the path strings p_i and the computation of the number of alternatives *n*. The specific part deals with the computation of the properties S_a and s_a.

The path strings are obtained by using the reduction rules from Table 2 and by using substitution. For example if

$$p_i \cong \{a; s;\} \text{ and } s \cong \langle b; c;\rangle$$

then

$$p_i \cong \{a; \langle b; c;\rangle\} = \langle\{a; b;\},\{ a; c;\}\rangle,$$

where the last (simplifying) equality may be checked by inspecting the tree (we slightly abused notation to avoid unnecessary notational complexity). Note that the simplification is necessary to obtain the alternative paths. The result of the substitution and simplification is shown in Fig. 14.

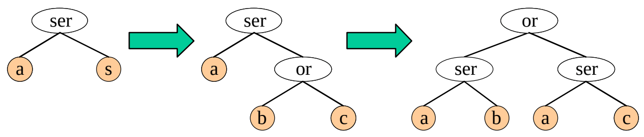

Fig. 14 Construction of alternative paths.

Clearly the aim of this step is, to get a single or-node at the root of the tree (a so-called *disjunctive normal form*). A similar reasoning may be used for other substitutions.

The number of alternative paths is computed as follows. Suppose we have two actions s_1 and s_2 which are a result of reduction steps. Furthermore, suppose that s_1 and s_2 correspond to n_1 and n_2 alternative paths, respectively. It is easily verified that the reduction step yields the number of alternatives as show in Table 3.

	# alternatives
ser_red(s_1, s_2)	$n_1 + n_2$
and_red(s_1, s_2)	$n_1 n_2$
or_red(s_1, s_2, p)	$n_1 + n_2$
rep_red(s_1, s_2 , p)	$2n_1 n_2$

Table 3 Number of alternatives.

The reduction algorithm is most easily implemented by using recursion, as described in [3].

Example 2

Consider the example process in Fig. 1. Duplicating actions, as explained in section 3.2, and using the reduction algorithm, we find the following five path strings:

1: {send;receive;inspect;[{renew;resend;},{list;store;}]}
2: {send;receive;inspect;
[{renew;*{detail;renew;}resend;},{list;store;}]}
3: {send;receive;inspect;
[{renew;resend;},{list;return;}]}
4: {send;receive;inspect;
[{renew;*{detail;renew;}resend;},{list;return;}]}
5: {send;receive;inspect;store;}

4.2 Examples

In this section we describe some typical examples of business process model analysis. In what follows we assume that actions s_1 and s_2 are obtained as a result of a reduction step. Let p_i denote an arbitrary path in the description of s_1 and p_j an arbitrary path in the description of s_2. Also, let s_i and s_j, respectively, denote the properties of these paths.

In a simple analysis the probabilities of the alternatives are computed.

"Probability of path"- analysis
Let the probability of p_i be given by p_i and of p_j by p_j . In the reduction step p_i and p_j are combined to a new path with probability p. The result of this combination is shown in Table 4.

	Probability p
$g_{\text{serial}}(p_i, p_j)$	$p_i\, p_j$
$g_{\text{and}}(p_i, p_j)$	$p_i\, p_j$
$(g_{\text{or},1}(p_i, p_j, p), g_{\text{or},2}(p_i, p_j, 1-p))$	$(p\, p_i ,(1-p)\, p_j)$
$(g_{\text{rep},1}(p_i, p_j, p), g_{\text{rep},2}(p_i, p_j, 1-p))$	$(p\, p_i ,(1-p)\, p_i\, p_j)$

Table 4 Probabilities of alternatives.

Example 2 (continued)

Using the probabilities in Fig. 2 we find the following probabilities for the five alternatives:

Path	1	2	3	4	5
Probability	0.09	0.01	0.09	0.01	0.80

The Testbed Studio tool, which supports the modelling and analysis of AMBER models, is able to show all paths. The path with the highest probability is shown in Fig. 15.

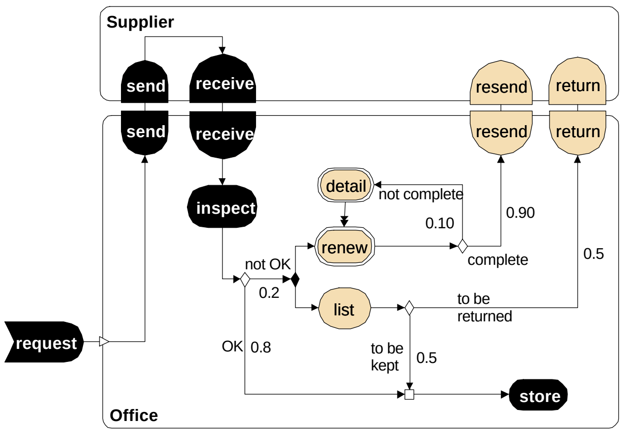

Fig. 15 Path with highest probability.

"Completion time of path"-analysis
If we assume that every action in the process model has a known probability distribution of its processing time, we may compute the probability distribution of the entire process. This we will call the completion time distribution of the process. This is described in detail in [3]. We can also compute the probability distribution of the completion time of each alternative path. This is what we will investigate here.

Suppose the paths p_i and p_i have probability density functions d_i and d_j, and corresponding distribution functions D_i and D_j respectively.
In the reduction step p_i and p_j are combined to a new path with probability density function d and corresponding distribution function D. The result of this combination is shown in Table 5.

	Probability function D
$g_{\text{serial}}(D_i, D_j)$	$D_i * D_j$
$g_{\text{and}}(D_i, D_j)$	$D_i D_j$
$(g_{\text{or},1}(D_i, D_j, p),$ $g_{\text{or},2}(D_i, D_j, 1-p))$	$(D_I * D_j , D_i * D_j)$
$(g_{\text{rep},1}(D_i, D_j, p),$ $g_{\text{rep},2}(D_i, D_j, 1-p))$	$(D_i , p\sum_{k=1}^{\infty}(1-p)^{k-1} D_i^{*(k+1)} D_j^{*k})$

Table 5 Completion time distributions of alternatives.

The $*$-symbol in the table denotes (discrete) convolution, i.e., for every discrete point in time

$$(D_i * D_j)(t) = \sum_{k=0}^{t} D_i(k) d_j(t-k)$$

and D^{*n} denotes repeated self-convolution:

$$D^{*0}\text{ (t)} = 1,\ D^{*1} = D,\ D^{*2} = D * D,\text{ etc.}$$

5. Conclusions and future work

In this paper we presented the AMBER language for business process modelling. This graphical modelling language has a formal mathematical basis, which makes it suitable for analysis purposes. Non-graphical information can be added to models by using profiles. For (basic and extended) SP models we showed that they have an equivalent graph representation. Analysis of these models is based on a tree reduction algorithm. Relaxing the SP constraint, also extended SP and approximate SP models can be analysed. The algorithm can easily be used for all kinds of business process analysis, such as (critical) path analysis and completion time analysis.
The algorithm can easily be adapted to other types of analysis, e.g. cost analysis, risk analysis, and fuzzy completion time analysis (based on [5]) on which we are currently working.

Acknowledgement
This paper is based on research from the Testbed project, which develops a language, methods and tools to support business process modelling and (re)design in the financial sector (see http://www.telin.nl/testbed/eng/default.html).

Project results have been applied in several real-life business cases. The project is a collaboration of the pension fund ABP, the Dutch Tax Department, ING Group, IBM and the Telematica Instituut, and is financially supported by the Dutch Ministry of Economic Affairs.

References:

[1] H.M. Franken, and W. Janssen, Get a Grip on Changing Business Processes: Results from the Testbed Project, *Knowledge & Process Management* (J. Wiley), 1998.

[2] W. Janssen, R. Mateescu, S. Mauw, P. Fennema, and P. van der Stappen, Model Checking for Managers, Proc. of the International SPIN Workshop, Toulouse, 1999.

[3] H. Jonkers, P. Boekhoudt, M. Rougoor, and E. Wierstra, Completion Time and Critical Path Analysis for the Optimisation of Business Process Models, *Proc. of the 1999 Summer Computer Simulation Symposium*, M.S. Obaidat, A. Nisanci and B. Sadoun (eds), Chicago, 1999, pp. 222-229.

[4] H. Jonkers, and M. van Swelm, Queueing Analysis to Support Distributed System Design, *Proc. of the 1999 Symposium on Performance Evaluation of Computer and Telecommunication Systems*, M.S. Obaidat and M. Ajmone Marsan (eds), 1999, pp. 300-307.

[5] A. Kaufmannn, and M.M. Gupta, *Introduction to Fuzzy Arithmetic*, Van Nostrand Reinhold Company, 1985.

[6] D.A.C. Quartel, L. Ferreira Pires, M.J. van Sinderen, H.M. Franken, and C.A. Vissers, On the Role of Basic Design Concepts in Behaviour Structuring, Computer Networks and ISDN Systems, Vol. 29, No. 29, 1997, pp. 413-436.

[7] R. Sahner, K.S. Trivedi, and A. Puliafito, Performance and Reliability Analysis of Computer Systems: an Example-Based Approach using the SHARPE Software Package, Kluwer Academic Publishers, 1996.