Modelling and analysing reliable service-oriented processes

Apostolos V. Zarras* and Panos Vassiliadis

Department of Computer Science,
University of Ioannina,
Ioannina, Greece
E-mail: zarras@cs.uoi.gr
E-mail: pvassil@cs.uoi.gr
*Corresponding author

Valerie Issarny

INRIA - UR de Rocquencourt,
Domaine de Voluceau, Rocquencourt, France
E-mail: Valerie.Issarny@inria.fr

Abstract: This paper introduces principled methods for the reliability analysis of business processes that rely on web services. The input to the problem is the BPEL specification of a business process and the output is the prediction of the process's reliability. The first step to this end involves a method for the translation of the BPEL specification to its corresponding UML model. The second step of the reliability analysis involves a principled way for the annotation of the UML model with the necessary extensions for the specification of reliability properties that characterise the behaviour of the elements that constitute the process. The third step of the analysis comprises the systematic mapping of the extended UML model to block diagrams and Markov models which are subsequently used to compute the reliability of the process.

Keywords: composite web services; BPEL; reliability.

Reference to this paper should be made as follows: Zarras, A.V., Vassiliadis, P. and Issarny, V. (2008) 'Modelling and analysing reliable service-oriented processes', *Int. J. Business Process Integration and Management*, Vol. 3, No. 3, pp.147–163.

Biographical notes: Apostolos Zarras received his PhD in Computer Science from the University of Rennes I, in 2000. He joined the Department of Computer Science of the University of Ioannina in 2004. His research interests include software engineering in general and in particular model-driven architecture development, quality analysis of software systems, middleware, service-oriented computing and pervasive computing. More information can be found at http://www.cs.uoi.gr/~zarras.

Panos Vassiliadis obtained his PhD from the Department of Electrical and Computer Engineering of the National Technical University of Athens (NTUA) in 2000. He has joined the Department of Computer Science of the University of Ioannina in 2002. So far, his research has focused on data warehousing, modelling of the architecture and evolution of databases and information systems, and web services. More information can be found at http://www.cs.uoi.gr/~pvassil.

Valerie Issarny received her PhD in Computer Science from the University of Rennes I, in November 1991, where she proposed an exception handling model for concurrent programming. During her PhD, she was a member of the LSP (Langages space operating system). From 1993 until 2001, she has been an INRIA Researcher in the Solidor group, examining solutions to the construction of robust and efficient distributed systems. Since 2002, she is a Research Director, leading the ARLES group at INRIA Rocquencourt, which investigates solutions to architecture-based development of ambient intelligence environments. More information can be found at http://www-rocq.inria.fr/arles/members/issarny.html.

1 Introduction

'How do we build large-scale enterprise information systems out of pre-existing elements and elements that are build from scratch?'

Nowadays, the web services architectural style appears as the most suitable answer to this question. Web services are web-enabled entities that confront the increased heterogeneity of software composition by encapsulating pre-existing elements and exporting them through platform-independent, standard interfaces to the rest of the information system. Interfaces are specified using the web services description language (WSDL) (W3C, 2001) and interaction is realised through XML messages that follow the SOAP standard (W3C, 2002). The orchestration of the business processes of systems that rely on the web service architectural style is specified using BPEL scenarios (IBM, 2002). These scenarios are executed on top of BPEL compliant execution engines.

'How do we assess the quality (e.g., reliability, availability, efficiency, etc.) of the business processes supported by the integrated system?' Regarding this question, the web services architectural style does not currently provide any particular support. This paper focuses in the reliability aspect and the primary goal is the provisioning of methods that enable the 'what-if' reliability analysis of complex BPEL business processes, with respect to various reliability properties (e.g., failure rate, redundancy, etc.) that may characterise the constituents of these processes. In other words, the refined question tackled in this paper is the following: 'given a BPEL scenario, how can we assess the risk of suffering failures during its execution?'

Consider the example of a pharmaceutical company that collects medicines and medical accessories from pharmaceutical factories and delivers them to pharmacies. Currently, whenever an order is placed by a client pharmacist, all the items of the order are assembled in a single package. A delivery company is employed to deliver the package to the client pharmacy. Still, some orders are much more urgent than the others and therefore the company wishes to extend its delivery process, by allowing the client to specify whether the items should be shipped in a piecemeal fashion, as soon as possible. The piecemeal delivery incurs a possible change to the business process of the company, which involves organising the delivery of the products through a dedicated delivery division, instead of resorting to the external delivery company.

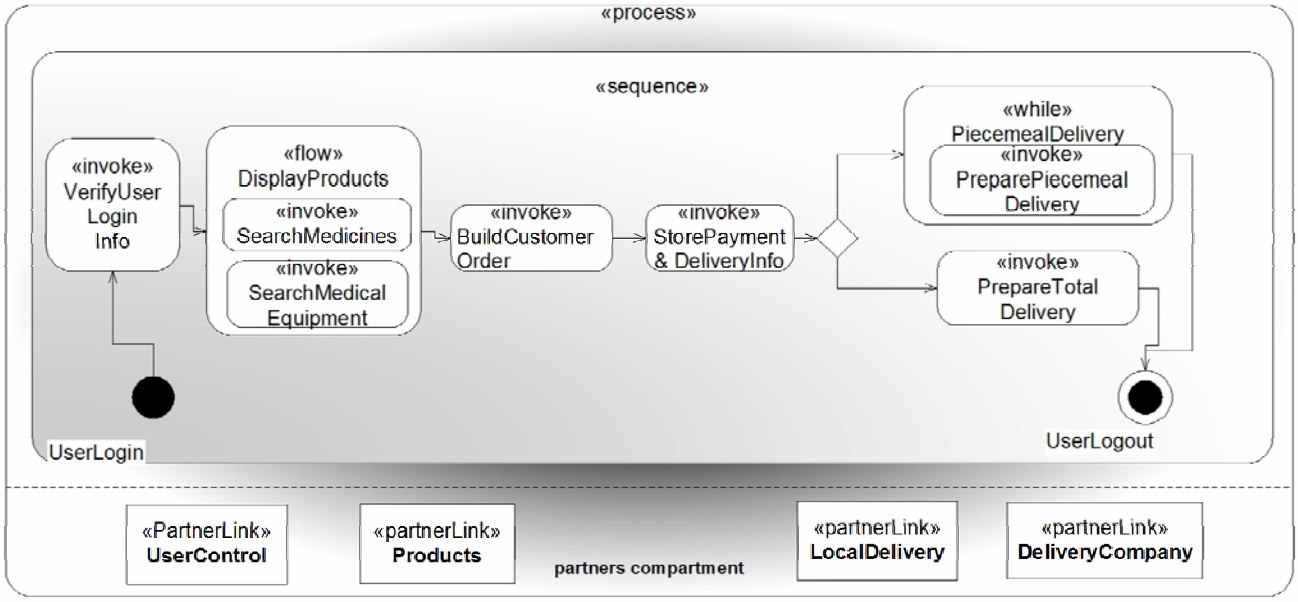
The back-stage process of the pharmaceutical company is fully automated; the integration of the various parts of the company's system with the external delivery company is facilitated through web services – see Figure 1. The pharmacists log-in, view medicines/equipment and compose an order. Then, the order is processed and a payment is made. The list of the ordered goods is passed to the delivery company's system for delivery. Once the pharmacy has received the goods, it confirms that the order has been well processed.

The manager of the company launches two workgroups to study the extension problem by providing a financial and an IT-expansion plan. The latter involves studying the risks and benefits of extending the current information system, by calculating development and maintenance efforts as well as the risk incurred by the new architecture. The potential drop of the reliability, after the extension, has a direct impact to the maintenance effort and the risk of the new system. Therefore, the IT team has to provide an assessment of how vulnerable the new system is going to be. The team is composed of experts familiarised with traditional modelling (e.g., UML) and reliability analysis (e.g., Markov models) techniques, but has no real experiences with novel technologies like BPEL. To proceed, the team decides to perform the following steps:

- map the BPEL specification of the company's business process to a familiar UML model
- enhance the UML model with reliability properties
- assess the reliability of the process through well-known techniques like block diagrams or Markov models.

Although techniques exist on similar problems, none of the above tasks is straightforward. A full technique to translate a BPEL to a UML model is currently not available. There is no standard method on how to annotate a BPEL-specific UML model with reliability properties, either. Moreover, there is no principled way to derive block diagrams or Markov models from BPEL-specific UML models. Finally, there is no clear winner on which reliability analysis technique to use for the assessment of the system's reliability – i.e., what technique should the IT team use given its constraints on time, resources and knowledge.

Figure 1 Reference example of a web service based business process



Based on the above discussion, the main contributions of this paper are as follows:

- A UML method is proposed for the specification of basic BPEL modelling constructs. This way, BPEL scenarios are mapped to UML activity models.
- The introduced UML constructs are associated with properties that serve for the reliability analysis of BPEL processes.
- Systematic methods are proposed for using the resulting, BPEL-specific UML models as inputs to two well-known reliability analysis techniques that rely on block diagrams and Markov models, respectively.
- The use of the aforementioned techniques is illustrated based on the reference example and their appropriateness is discussed with respect to their precision and the resources they require for the reliability analysis of business processes.

The work proposed in this paper extends previous work presented in Zarras et al. (2004). Specifically, the UML modelling method proposed in Zarras et al. (2004) is aligned with the standard UML profile for modelling quality of service and fault tolerance characteristics and mechanisms (OMG, 2004). Moreover, the proposed methods are generalised for mapping BPEL-specific UML models to traditional reliability analysis models to account for various kinds of faults and failures. Last but not least, care is taken for different possible execution scenarios that may originate from a BPEL process specification.

The remainder of this paper is structured as follows. Section 2 details the proposed UML modelling method and the association of BPEL-specific UML constructs with reliability properties. Sections 3 and 4 detail the mapping of BPEL-specific UML models to block diagrams and Markov models. The use of these techniques is illustrated in Section 5. Section 6 discusses the related work. Finally, Section 7 summarises the contribution of this paper and discusses future work.

2 Specifying business processes in UML

Mapping BPEL specifications in UML, consists of defining a set of stereotypes that represent BPEL constructs (Section 2.1). The method is straightforward: the fundamental BPEL constructs are exhaustively enumerated (first column of Table 1) and mapped to the appropriate UML constructs (second column of Table 1). In Mantell (2003), there has been a similar attempt to achieve this goal. However, the mappings at the level of BPEL concept categories (e.g., BPEL activities, BPEL variables, etc.) were discussed. Moreover, the proposed approach goes one step further by associating the proposed stereotypes with properties that serve for the reliability analysis (Section 2.2). These properties are defined in accordance with the standard UML profile for modelling quality of service and fault tolerance characteristics and mechanisms (OMG, 2004).

2.1 Specifying BPEL constructs

A business process in BPEL is described in terms of a *process*, specified using a homonymous stereotype (Table 1). The process consists of *activities*, specified using the *activity* stereotype. The execution of an activity relies on the use of an interface that is provided by a basic web service. This interface is termed partner in BPEL and it is modelled using the *partnerLink* stereotype.

Table 1 Stereotypes for structuring composite web services

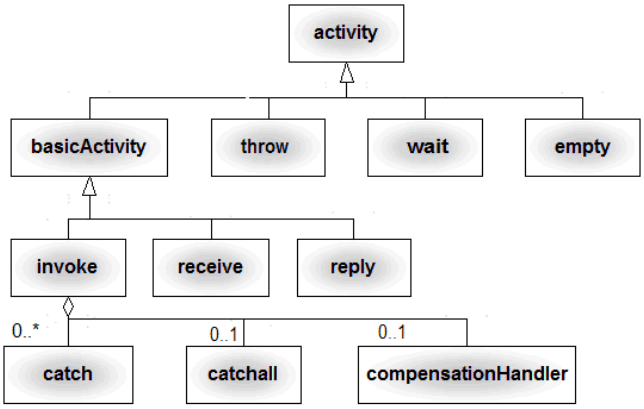
Stereotype	*UML base class*	*Parent*
process	Activity	NA
activity	ExecutableNode	NA
partnerLink	ObjectNode	NA
variable	DataStoreNode	NA
catchAll, catch	ExceptionHandler	NA
onMessage, onAlarm	AcceptEventAction	activity
compensationHandler	ExceptionHandler	NA
basicActivity	ExecutableNode	activity
invoke	CallAction	basicActivity
receive	AcceptCallAction	basicActivity
reply	ReplyAction	basicActivity
throw	RaiseExceptionAction	activity
wait	AcceptEventAction	activity
empty	Action	NA
sequence, pick, flow	ActivityPartition	activity
switch	DecisionNode	activity
while	LoopNode	activity

The process specification further includes the description of *fault* and *event* handlers (specified using the *faultHandler* and the *eventHandler* stereotypes). A fault handler consists of an activity, triggered upon the occurrence of a failure. By definition, 'failure means deviation from compliance with the system specification' (Laprie, 1985). Including, thus, a fault handler in the specification of a process implies that the occurrence of the respective failure does not cause deviation from the system specification. Hence, faults that are properly handled are not considered in the analysis.

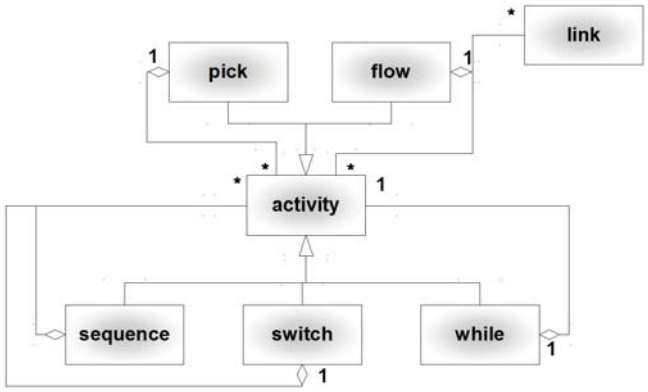
The proposed representation allows specifying different kinds of *basic* [Figure 2(a)] and *structured* [Figure 2(b)] BPEL activities. The execution of a basic activity relies on a single web service. On the other hand, a structured activity consists of a set of (basic or structured) activities and prescribes the order of their execution. In other words, it defines a number of control and data flow dependencies, specified using standard ControlFlow (i.e., arrows stating that the target activity is triggered when the execution of the source activity is done) and Dataflow (i.e., arrows stating that the target activity accepts input from the source activity) elements. The different kinds of basic activities supported are:

- *invoke* activities, specifying the synchronous or asynchronous invocation of a web service operation.
- *receive* activities, describing the reception of SOAP messages. A receive activity may be an initial activity of the process. In this case, any other activities that precede it or execute simultaneously must also be initiating receive activities.
- *reply* activities, delineating responses to SOAP messages that were previously received during the execution of receive activities.

Figure 2 Different kinds of (a) basic activities and (b) structured activities



(a)



(b)

The different kinds of structured activities supported by the proposed representation are:

- *sequence* activities, consisting of activities that execute sequentially.
- *switch* activities, consisting of ordered activities associated with conditions. During a switch activity, only the first activity whose condition evaluates to true actually executes.
- *while* activities, comprising a single activity that executes for a number of times.
- *pick* activities, consisting of one or more event handlers.
- *flow* activities, comprising one or more activities, which by default execute concurrently. However, there may exist control and data flow dependencies between them, imposing a certain execution order.

As already mentioned, the method for mapping BPEL to UML modelling elements is straightforward: the fundamental BPEL constructs (first column of Table 1) are exhaustively enumerated and mapped to the appropriate UML constructs (second column of Table 1). Despite the simplicity of the method, there are a couple of issues to be highlighted. A first comment concerns the recursive application of the method: the proposed mapping method can be recursively applied in the case of BPEL invoke activities that correspond to nested BPEL scripts, resolving each BPEL workflow one at a time. A second remark to be made is that the formal underpinnings of the BPEL-to-UML mapping are well beyond the scope of the paper, which focuses on reliability analysis. Despite the significance of the topic, providing the formal semantics to prove the correctness of the mapping process is not dealt with in this paper. Nevertheless, this issue is an interesting point for future research.

Figure 1 illustrates the use of the stereotypes defined in this section for the specification of the reference example. Specifically, observe the use of a switch activity for the specification of the two alternative ways of delivery (i.e., the `PiecemealDelivery` while activity that executes using the `LocalDelivery` partner and the `PrepareTotalDelivery` invoke activity that executes using the `DeliveryCompany` partner).

2.2 Specifying reliability properties

The basic properties that characterise the stereotypes defined in Section 2.1 are given in Table 2. As imposed by the standard UML profile for modelling quality of service and fault tolerance characteristics and mechanisms (OMG, 2004), they are specified as QoS characteristics, consisting of different QoS dimensions.

Specifically, the process stereotype is associated with the *reliability* characteristic that has a single dimension, *prob*, defined as the probability that the process executes correctly for a given time duration. Time is also a reliability characteristic of the process stereotype.

The impairments to reliability considered in the analysis are the faults and the failures of the partners, used by the activities of the process. Therefore, the partnerLink stereotype is associated with *fault* and *failure* characteristics. Faults appear with a certain rate, specified using the *failure-rate* dimension (Table 2). Faults and failures are characterised by further dimensions (e.g., the *nature* of faults, the *persistence* of faults, etc.), allowing to distinguish between different kinds of them (Table 3). For instance, *physical faults* are permanent faults that relate to physical phenomena and not to the partner's internal or external condition. On the other hand, *transient faults* are temporary external faults, resulting from the interaction of the partner with the environment. Transient faults disappear with a certain rate (specified using the *disappearance-rate* dimension, given in Table 2). *Intermittent faults* are temporary internal faults, resulting from the interference between the different parts of the partner. Intermittent faults may be either active or benign. In the former case, the failed partner provides incorrect operations, while in the latter, the previous does not hold. Intermittent faults repeatedly go from active to benign and back to active with certain rates (specified using the *active-to-benign-rate* and the *benign-to-active-rate* dimensions, defined in Table 2) (Butler, 1992). More detailed definitions of the various fault and failure dimensions of Table 2 can be found in Laprie (1985).

According to the standard UML profile for modelling quality of service and fault tolerance characteristics and mechanisms (OMG, 2004), more than one partners may be associated with the same fault and failure characteristics. Hence, it is possible to specify fault and failure dependencies between different partners. Such dependencies may result from several reasons such as a common development process that was followed for the partners or a common platform (Eckhardt and Lee, 1985; Knight and Leveson, 1986). It should be further noted that the faults and the failures of the underlying BPEL execution engine that executes business processes may also be considered as impairments to the reliability of these processes (Issarny et al., 2002). Dealing with the aforementioned issue is out of the scope of this paper. However, it would imply associating the stereotypes defined for basic and structured activities with reliability characteristics that are similar with the ones defined for the partnerLink stereotype.

The partnerLink stereotype is further associated with the fault tolerance characteristic, which consists of different dimensions, prescribing the fault tolerance technique that may be used for a partner (Laprie et al., 1990). A partner may represent a redundancy schema, i.e., a configuration of redundant partners, which behave as a single fault tolerant unit. The schema is characterised by the error detection mechanism used, the number of partners ($no_{partners}$) that constitute it, the number of partner failures that can be tolerated ($no_{failures}$), etc. At this point, it should be noted that it is the responsibility of the designer to specify the aforementioned properties in a way that reflects the potential existence of dependent failures (Eckhardt and Lee, 1985; Knight and Leveson, 1986). If, for instance, a schema comprises five partners and two of them fail in a dependent manner, then the actual value of the $no_{partners}$ property should be four. In general, if a partner represents a redundancy schema (i.e., it is associated with the fault tolerance characteristic), then it can be associated with $no_{partners}$ fault and failure characteristics, one for every constituent of the schema.

Table 2 Properties of the UML stereotypes

Stereotype	*QoS dependability characteristics*					
process	*reliability*					
	QoS dimensions					
	prob	0..1				
partnerLink	*fault*		*failure*		*fault-tolerance*	
	Qos dimensions		*QoS dimensions*		*QoS dimensions*	
	nature	{accidental, intentional}	domain	{time, value}	error-detection	{vote, comp, acceptance}
	cause	{physical, human}	perception	{consistent, inconsistent}	execution	{parallel, sequential}
	phase	{design, operational}			confidence	{absolute, relative}
	persistence	{permanent, temporary}			service-delivery	{continuous, suspended}
	failure-rate	Real			$no_{partners}$	Integer
	disappearance-rate	Real			$no_{failures}$	Integer
	active-to-benign-rate	Real				
	benign-to-active-rate	Real				
	Service-delivery	{continuous, suspended}				
basicActivity	*completion*					
	QoS dimensions					
	completion-rate	Real				
	mean-completion-time	Real				
	compl-dev	Real				
while	*iterations*					
	QoS dimensions					
	no-iter	Integer				
switch, pick	*Branches*					
	QoS dimensions					
	branch_prob	Real				

In the reliability analysis, the *mean-completion-time* and deviation (or the *completion-rate*) of basic activities is further taken into account. The completion time of structured activities is a function of the completion times of the basic activities that constitute them. While activities are associated with a characteristic that represents the (possibly approximate) number of iterations performed by these activities. Switch and pick activities are associated with arrays of real values, ranging from zero to one (i.e., the *branch-prob* property). Each value represents the probability of executing one of the constituents of these activities.

An example of specifying reliability properties is given in Figure 3. Specifically, the dimensions of the fault and the failure characteristics are specified for the `DeliveryCompany` partner.

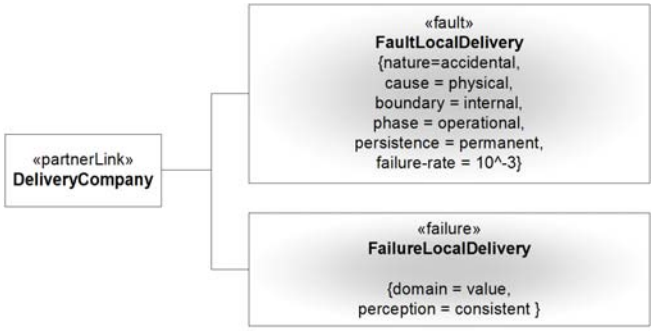
Table 3 Different classes of faults

Nature		*Cause*		*Boundary*		*Class of faults*
Accidental	*Intentional*	*Physical*	*Human*	*Internal*	*External*	
x		x		x		Physical faults
x		x			x	
x		x			x	Transient faults
x		x		x		Intermittent faults
x			x	x		
x			x	x		Design faults
x			x		x	Interaction faults
	x		x	x		Malicious logic
	x		x	x		
	x		x		x	Intrusions
	x		x		x	

Phase		*Persistence*		*Class of faults*
Design	*Operational*	*Permanent*	*Temporary*	
	x	x		Physical faults
	x	x		
	x		x	Transient faults
	x		x	Intermittent faults
x			x	
x		x		Design faults
	x		x	Interaction faults
x		x		Malicious logic
x			x	
	x	x		Intrusions
	x		x	

Source: Defined in Laprie (1985)

Figure 3 Reliability properties for the DeliveryCompany partner



3 Block diagrams for business processes

In principle, a block diagram is used to represent a constraint for correctly executing a process. The block diagram consists of blocks (i.e., boxes), representing the partners that provide the basic web services, used in the process. Those blocks are connected using serial connections. More specifically, for every structured activity A of the process, consisting of the $\alpha_1, \alpha_2, \ldots \alpha_N$ constituent activities, we have:

1 If A is a sequence, a flow or a while activity, all of the constituent activities are needed to successfully complete A (for while activities $N = 1$).

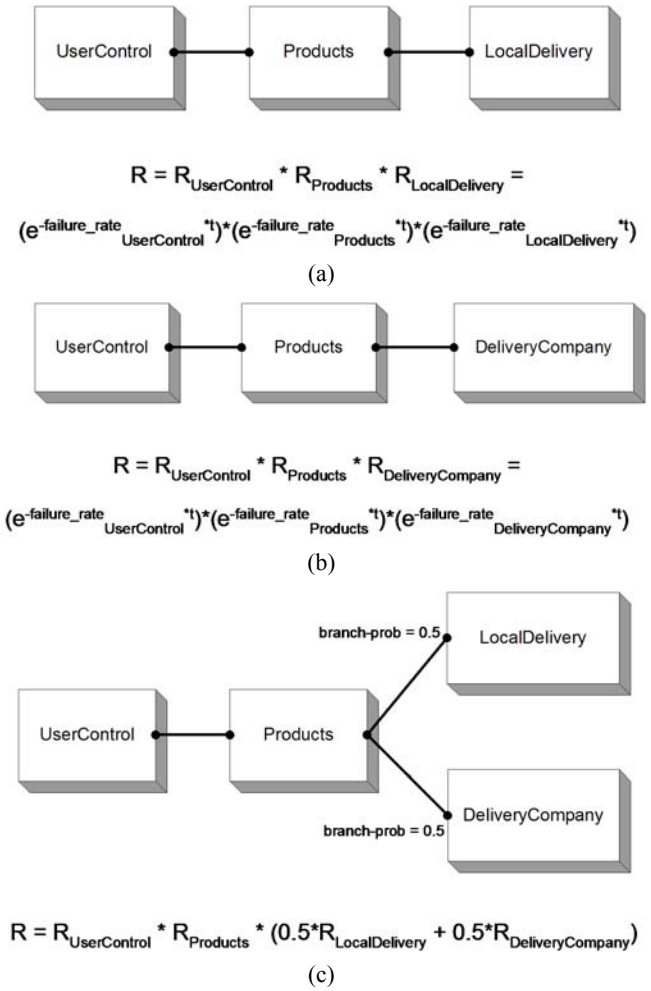
2 If A is a switch or a pick activity with N branches, any constituent activity may execute, depending on the switch condition or the particular events that occur at runtime. Hence, A implies the existence of N possible execution paths. Some of these paths may involve the use of different sets of partners leading into different block diagrams. At this point, the purpose of the reliability analysis may differ depending on the preferences of the designer. In the case that the designer performs a what-if analysis, each path should be analysed as a different solution, independently from the others. In the case that the designer is interested on the impact of each choice on the reliability of the overall process, different probabilities for the choice of each path must be assigned. In the absence of any extra information (or input by the designer), the former is considered as the default choice. Based on the previous,

in the systematic block diagram construction, two possible alternatives may be followed:

- Either we generate all the different block diagrams that result from A.
- Or we assume a certain probability $branch-prob_{\alpha_i}, i=1,\ldots N$ for every branch of A and construct a single block diagram that models all the paths. The branch probabilities may be identified based on experimental results derived from monitoring the process execution.

Taking the reference example, we get three different block diagrams. Figures 4(a) and 4(b) give, respectively, the block diagrams that result from the different alternative delivery options offered by the process, while Figure 4(c) gives the overall weighted block diagram.

Figure 4 Block diagrams for the reference example, (a) block diagram for local delivery (b) block diagram for external delivery (c) block diagram for both delivery alternatives



3 Based on 1–2, the block diagram is constructed as follows:

a For the basic activities $\alpha_1, \alpha_2, \ldots \alpha_L$ of $A(L < N)$:

- Select the $p_1, p_2, \ldots p_K$ non-fault-tolerant partners (i.e., the partners that do not represent redundancy schemas), used by $\alpha_1, \alpha_2, \ldots \alpha_L$. Create a new block for every such partner. The blocks are connected with serial connections.
- Select the $p_{K+1}, p_{K+2}, \ldots p_M$ fault tolerant partners, used by $\alpha_1, \alpha_2, \ldots \alpha_L$. For every such partner, create a (i-out-of-$no_{partners}$) parallel connection that connects $no_{partners}$ blocks, representing the schema constituents. To correctly execute the process, $i = no_{partners} - no_{failures}$ of the schema constituents must be operational.

b For the *composite activities* $\alpha_{L+1}, \alpha_{L+2}, \ldots \alpha_N$ of A recursively follow 1–3.

To calculate, the reliability value from the block diagram specification, the reliability values that characterise the individual blocks and parallel connections are multiplied. The reliability value for a block is calculated in terms of the failure-rate characteristic of the partner that is represented by the block (e.g., Figure 4).

4 Markov models for business processes

The systematic construction of Markov models for BPEL processes is more complicated compared to the case of block diagrams. In this section, the issue of modelling BPEL processes with Markov models is discussed first. Then, the systematic specification of such models is detailed. The proposed approach is built upon a generic framework for the generation of Markov models, which has been proposed in Johnson (1988). Specifically, this work is adapted to the specificities introduced by BPEL processes.

4.1 Markov models for BPEL processes

A Markov model for a BPEL process consists of a set of transitions between states of the process. A state describes a situation where either the process executes correctly or not. In the latter case, the process is in a *death-state*. The state of the process depends on the situation of its basic activities (which may be encapsulated in structured activities or not) and the situation of the partners, used for executing these activities. The structured activities that encapsulate basic activities do not directly affect the situation of a process as they are not involved in performing any serious computation. Switch and pick activities are treated similarly to the case of block diagrams, i.e., either we specify different Markov models for different process execution paths or we specify an overall Markov model, using the branching probabilities associated to these kinds of activities. Flow and sequence activities serve as a structuring mechanism, which determines the execution order of their encapsulated activities. Accounting for failures of the middleware mechanisms that coordinate the

aforementioned execution order further complicates the reliability analysis and is out of the scope of this paper. Previous work presented in Issarny et al. (2002) tackles this particular problem in the case of conventional composite systems and may also serve in the case of systems that rely on the web services architectural style.

Hence, the state of a process is modelled as a composition of sub-states, representing the partners and the basic activities of the process. A basic activity may be in four different states: *inactive*, *active*, *complete* or *failed*. Later, the special case of activities encapsulated in while activities is further discussed. The range of the different state situations for a partner depends on the partner's faults. A partner with a physical or a transient fault may be: *operational* or *failed*. Similarly, a partner with an intermittent fault may be: *operational*, *failed-active* or *failed-benign*. The range of state situations for a partner that represents a redundancy schema further depends on the number of schema constituents, the number of failures that can be tolerated and the possible existence of dependent failures. For instance, for a redundancy schema of five partners and two failures, we have four possible situations in the absence of dependent failures: *all partners are operational*, *one partner is failed*, *two partners are failed* and *three partners are failed (death-state)*.

In the reference example, suppose that the `DeliveryCompany` partner fails because of a physical fault. Then, part of the Markov model that describes the execution path that involves the `DeliveryCompany` partner is given in Table 4 [the format used to present the Markov model is the one assumed by the SURE reliability analysis tool (Butler, 1992), which is used for solving Markov models]. The state of the process is modelled by a tuple of 11 integer values representing the sub-states that correspond to the three partners and the eight basic activities of the process. The first value of the tuples is zero in states where the `UserControl` partner is operational and one in states where it is failed. For the activities (the last eight values of each tuple in Table 4), the values of the tuples are 0, 1, 2 or 3, in states where the activities are *inactive*, *active*, *complete* or *failed*, respectively.

The Markov model comprises the following different kinds of transitions:

- *Transitions for partner failures*, which model partner's failures that do not affect the activities which use these partners because the failures take place before the beginning of their execution. These transitions are characterised by the failure-rate or the benign-to-active-rate of the partners (Table 2).
- *Transitions for partner recovery*, which model the recovery of partners that previously failed. These transitions are characterised by the disappearance-rate or the active-to-benign-rate of the partners (Table 2).
- *Transitions for activity activation*, which take the process from a state where an activity α is inactive, to a state where the activity executes (e.g., transition 3 in Table 4). These transitions take place only if the activities upon which α depends are complete and the partner used by α is not failed. These transitions are characterised by a default mean-time for triggering activities.
- *Transitions for activity completion*, taking the process from a state where an activity α is active, to a state where α is complete (e.g., transition 4 in Table 4). These transitions are characterised by the mean-completion-time of the activity (Table 2).
- *Transitions for activity failure*, modelling the failure of activities that already execute or the failure of activities that are ready to execute (an activity α is ready to execute when all the activities upon which it depends are complete; at this time, α is going to be triggered). The activities fail due to the failure of the partners used by these activities. Therefore, the transitions are characterised by the failure-rate or the benign-to-active-rate of the failed partners (Table 2).

The overall reliability of a process is calculated with respect to the probability of reaching a death-state of the Markov model within a given time duration t. The calculation of this value involves solving a system of first-order differential equations (Butler, 1992).

Table 4 Markov model for the reference example

Transition	*Source*		*Target*		*Rate*
1	3	(* 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 *)	4	(* 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 *)	<mean-trig-time, trig-dev>
2	4	(* 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 *)	5	(* 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0 *)	<mean-completion-time, compl-dev>
3	5	(* 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0 *)	6	(* 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0 *)	<mean-trig-time, trig-dev>
4	6	(* 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0 *)	7	(* 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0 *)	<mean-completion-time, compl-dev>
5	6	(* 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0 *)	1	(* 1, 0, 0, 2, 3, 0, 0, 0, 0, 0, 0 DEATH *)	failure-rate
6	7	(* 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0 *)	8	(* 0, 0, 0, 2, 2, 1, 1, 0, 0, 0, 0 *)	<mean-trig-time, trig-dev>
7	8	(* 0, 0, 0, 2, 2, 1, 1, 0, 0, 0, 0 *)	9	(* 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0 *)	<mean-completion-time, compl-dev>
8	8	(* 0, 0, 0, 2, 2, 1, 1, 0, 0, 0, 0 *)	1	(* 0, 1, 0, 2, 2, 3, 3, 0, 0, 0, 0 DEATH *)	failure-rate
9	9	(* 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0 *)	10	(* 0, 0, 0, 2, 2, 2, 2, 1, 0, 0, 0 *)	<mean-trig-time, trig-dev>
10	10	(* 0, 0, 0, 2, 2, 2, 2, 1, 0, 0, 0 *)	11	(* 0, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0 *)	<mean-completion-time, compl-dev>
11	10	(* 0, 0, 0, 2, 2, 2, 2, 1, 0, 0, 0 *)	1	(* 0, 1, 0, 2, 2, 2, 2, 3, 0, 0, 0 DEATH *)	failure-rate
12	11	(* 0, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0 *)	12	(* 0, 0, 0, 2, 2, 2, 2, 2, 1, 0, 0 *)	<mean-trig-time, trig-dev>
13	12	(* 0, 0, 0, 2, 2, 2, 2, 2, 1, 0, 0 *)	13	(* 0, 0, 0, 2, 2, 2, 2, 2, 2, 0, 0 *)	<mean-completion-time, compl-dev>
14	12	(* 0, 0, 0, 2, 2, 2, 2, 2, 1, 0, 0 *)	1	(* 0, 1, 0, 2, 2, 2, 2, 2, 3, 0, 0 DEATH *)	failure-rate
15	13	(* 0, 0, 0, 2, 2, 2, 2, 2, 2, 0, 0 *)	14	(* 0, 0, 0, 2, 2, 2, 2, 2, 2, 1, 0 *)	<mean-trig-time, trig-dev>
16	14	(* 0, 0, 0, 2, 2, 2, 2, 2, 2, 1, 0 *)	15	(* 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0 *)	<mean-completion-time, compl-dev>
17	14	(* 0, 0, 0, 2, 2, 2, 2, 2, 2, 1, 0 *)	1	(* 0, 0, 1, 2, 2, 2, 2, 2, 2, 3, 0 DEATH *)	failure-rate
18	15	(* 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0 *)	16	(* 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 1 *)	<mean-trig-time, trig-dev>
19	16	(* 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 1 *)	2	(* 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2 *)	<mean-completion-time, compl-dev>

Notes: failure-rate = 10–3; Mean-trig-time = 0.005; Trig-dev = 0; Mean-completion-time = 4; and Compl-dev = 0

4.2 The general framework for the specification of Markov models

Generally, it is recognised that the specification of Markov models is a complex and error-prone task (Johnson, 1988). To deal with this problem, Johnson (1988) proposed an algorithm that relies on the concepts discussed in the previous subsection. In particular, states are modelled as shown in the example of Table 4. The algorithm generates a Markov model, given the following input:

- *The definition of the range of states for the Markov model*. The range definition is given as a tuple of integer variables. Each variable represents the range of all possible state situations for a modelled element (e.g., a partner, an activity, etc.).
- *The definition of a death-state constraint for the Markov model*, i.e., a conditional statement, defined on the values of the range variables. This statement evaluates the true for tuples that represent the death-states of the Markov model.
- The definition of an initial state for the Markov model.

The algorithm further accepts as input *transition rules* between sets of Markov states. A transition rule consists of a *conditional statement* and a *transition statement*. The conditional statement is defined on the values of the range variables and identifies a set of source states that have common features (e.g., states where a particular partner p is operational). From all these source states, there should be transitions to target states, which also share common features (e.g., p is failed in all target states). Moreover, the transitions to the target states are characterised by a common rate (e.g., the failure-rate that characterises p). The transition statement of the rule specifies this common rate and the common features shared amongst the set of the target states.

Given the above, the algorithm starts from the initial Markov state and recursively applies the transition rules, as long as, their conditional statements hold. During a recursive step (for a particular transition rule), the algorithm produces a transition to a state derived from the initial one. If the death-state constraint holds for the resulting target state, the recursion stops. That way, the algorithm automatically produces all the possible state transitions for the Markov model.

Taking the particular execution of the reference example that involves the `DeliveryCompany`, the state range definition is as follows:

space = (

UserControl	0..1,
Products	0..1,
DeliveryCompany	0..1,
UserLogin	0..3,
VerifyUserLoginInfo	0..3,
SearchMedicalEquipment	0..3,
SearchMedicines	0..3,
BuildCustomerOrder	0..3,
StorePaymentInfo	0..3,
PrepareTotalDelivery	0..3,
UserLogout	0..3

);

Moreover, the death-state constraint for the same process is:

deathif ($VerifyUserLoginInfo = 3 \vee$
$SearchMedicalEquipment = 3 \vee SearchMedicines = 3 \vee$
$BuildCustomerOrder = 3 \vee StorePaymentInfo = 3 \vee$
$PrepareTotalDelivery = 3$)

This constraint states that every tuple with $VerifyUserLoginInfo = 3$ represents a death-state (w.r.t., the modelling of activity failures in the previous subsection). Similarly, every tuple with $SearchMedicines = 3$ also represents a death-state. An example of a simple transition rule for the reference example is the following:

if $\underbrace{DeliveryCompany = 0 \wedge PrepareTotalOrder = 1}_{\text{conditional statement}}$ then
tranto (
$\ldots \underbrace{DeliveryCompany + 1, \ldots\ PrepareTotalOrder + 2}_{\text{common features for the target states}}$
) by 0.01;
endif;

The above rule refers to source states that share the following features:

- the `DeliveryCompany` partner is operational ($DeliveryCompany = 0$)
- the `PrepareTotalOrder` activity is active ($PrepareTotalOrder = 1$).

According to the transition statement of the rule, for every source state, there exists a transition to a target state and the common features of all target states are:

- the `DeliveryCompany` partner is failed ($DeliveryCompany = 1$)
- the `PrepareTotalOrder` activity is failed because of the customer failure ($PrepareTotalOrder = 3$).

The rate that characterises all of the transitions prescribed by the above rule is 0.01; that is the failure-rate of the `DeliveryCompany` partner.

Still, the specification of transition rules is a complicated task. To alleviate this problem, this paper proposes a *systematic method for specifying input models for Johnson's (1988) algorithm, from BPEL-specific UML models*. The generated models serve as input to the ASSIST tool (Johnson and Boerschlein, 2000) that implements Johnson's algorithm and generates complete Markov models. Finally, the Markov models can be given as input to the SURE analysis tool (Butler, 1992), which calculates corresponding reliability values.

4.3 Generating state-range definitions

The generation of a state range definition from the BPEL-specific UML model of a business process relies on the following steps:

- First, we select all the partners of the process that do not represent a redundancy schema. For each one of them, a corresponding variable is created in the state-range definition. The range of the integer values for this variable depends on the fault and the failure properties that are associated with the partner. More specifically, we have:
  - For a partner p with physical, transient, design, interaction, permanent intrusion or permanent malicious logic faults (Table 3), the value of the variable is zero in states where p is operational and one in states where p is failed.
  - For a partner p with intermittent, temporary intrusion or temporary malicious logic faults (Table 3), the value of the variable is zero in states where p is operational, one in states where p is failed-active and two in states where p is failed-benign.
- Then, we select all the partners that represent a redundancy schema. Each one of them, p, consists of $no_{partner}$ redundant partners and may tolerate $no_{failure}$ failures.
  - If p is a schema with physical, transient, design, interaction, permanent intrusion or permanent malicious logic faults, we create an integer variable that ranges from zero, in states where all partners are operational, to $no_{failure} + 1$, in states where the number of failed partners exceeds the number of failures that can be tolerated.

- If p is a schema with intermittent, intrusion or temporary malicious logic faults, we create two integer variables. The first one ranges from zero to $no_{failure}+1$ and represents the number of failed partners in the schema. The second variable ranges from zero to $no_{failure}$ and represents the number of failed-benign partners in the schema.

- Following, we select all the basic activities that are specified in the process (some of them may be encapsulated into structured activities). For each activity α, we create a variable in the state-range definition. More specifically, if α is encapsulated in a while activity that performs $no\text{-}iter$ iterations, the variable takes values from zero to $no\text{-}iter+1$. Otherwise, the variable takes values from zero to three. The semantics of these values are summarised in Table 5.

Table 5 Range of state-range variables for activities

Embedded in while		*Not embedded in while*	
Value	*Semantics*	*Value*	*Semantics*
0	α is inactive	0	α is inactive
i:1,…, $no\text{-}iter-1$	α is executed for the ith time	1	α is active
$no\text{-}iter$	α is complete	2	α is complete
$no\text{-}iter+1$	α is failed due to the failure of the partner used by this activity	3	α is failed due to the failure of the partner used by this activity

Based on the previous steps, the state-range definition for the execution path of the reference example that involves the `LocalDelivery` partner is:

space = (

UserControl	0..1,
Products	0..1,
LocalDelivery	0..1,
UserLogin	0..3,
VerifyUserLoginInfo	0..3,
SearchMedicalEquipment	0..3,
SearchMedicines	0..3,
BuildCustomerOrder	0..3,
StorePaymentInfo	0..3,
PreparePiecemealDelivery	0..6,
UserLogout	0..3

);

In the previous definition, it is assumed that the customer ordered five different products. Therefore, the `PreparePiecemealDelivery` ranges from zero to six. The latter is the only difference between this state-range definition and the one that was given in Section 4.2 for the execution path that involves the `DeliveryCompany` partner.

4.4 Generating death-state constraints

A process is considered as failed in states where any of its basic activities is failed (more precisely, failures of initial receive activities are not considered because in such cases, the overall process is not even initiated). Hence, to generate a death-state constraint, all the basic activities are selected. Then, the death-state constraint is built as the disjunction of a number of Boolean expressions. Each expression involves a state-range variable that represents one of the activities, say α. If α is encapsulated in a while activity that performs $no\text{-}iter$ iterations, the expression evaluates to true if the variable equals to $no\text{-}iter+1$ (Table 5). In all other cases, the expression evaluates to true if the variable equals to three (Table 5).

In the execution path of the reference example that involves the `LocalDelivery` partner, the death-state constraint is:

deathif ($VerifyUserLoginInfo=3 \vee$
$SearchMedicalEquipment=3 \vee SearchMedicines=3 \vee$
$BuildCustomerOrder=3 \vee StorePaymentInfo=3 \vee$
$PrepareTotalDelivery=6$)

4.5 Generating transition rules

In Section 4.1, four different categories of transitions were identified. Consequently, here, four different categories of transition rules are discussed.

- *Transition rules for partner failures:* Transition rules for partner failures are specified for all partners, independently from the classes of faults that characterise them. For every partner that does not represent a redundancy schema, a rule is specified whose conditional statement holds for all source states where the partner is operational and the activities that use the partner are inactive. The rule prescribes that for these source states, there should be transitions to target states, whose common feature is that the partner is failed. The rate for these transitions is the failure-rate or the benign-to-active-rate of the partner. If the partner represents a redundancy schema, the conditional statement holds for all source states where the number of failures that occurred is less or equal to the number of failures, $no_{failure}$, that can be tolerated. For all these source states, there should be transitions to target states, whose common feature is that the number of failures is increased by one.

Following, an example of a rule is given for the `UserControl` partner of the reference example:

if $UserControl = 0 \wedge VerifyUserLoginInfo = 0$ then
  tranto ($UserControl = 1$) by failure-rate;
endif;

- *Transition rules for partner recovery:* These rules are specified for partners with temporary faults. Specifically, for every partner that does not represent a redundancy schema, a transition rule is specified. If the partner is characterised by a (temporary) *external* fault (i.e., transient, interaction or intrusion fault), then the conditional statement of the generated rule holds for states where the partner is failed. The transition statement specifies transitions to target states, whose common feature is that the partner is operational again. The rate for these transitions is the disappearance-rate, associated with the partner. Similarly, if the partner is characterised by a (temporary) *internal* fault (i.e., intermittent or temporary malicious logic fault), then the conditional statement of the rule holds for source states where the partner is failed and the fault is active. The transition statement prescribes transitions to target states, whose common feature is that the partner is still failed, but the fault is benign. The rate for these transitions is the active-to-benign-rate, associated with the partner. For partners that represent redundancy schemas, similar rules are specified.

  Below, an example of a rule is given, for the `UserControl` partner, if we suppose that it fails because of a transient fault:

  if $UserControl = 1$ then
    tranto ($UserContol = 0$) by disappearance-rate;
  endif;

- *Transition rules for activity activation:* For every basic activity α of the process, a transition rule is generated, whose conditional statement holds for states where:

  1 the activity is inactive
  2 the activities upon which α depends are complete
  3 the partner, used by α is operational.

  Hence, the conditional statement is built based on the dataflow and control dependencies, specified for α. The transition statement of the rule states that for all source states, there should be transitions to target states, whose common feature is that α is active.

  If α is embedded in a pick or a switch activity and we generate an overall Markov model for the process, then the generated transitions are characterised by the branch-prob of the corresponding branch. If α is encapsulated in a flow activity together with activities, $\beta, \gamma, \ldots$, then the conditional and the transition statements of the rule involve all these activities, which are concurrently activated.

  Below, an example of a rule is given for the activation of the activities that are included in the `DisplayProducts` flow of the reference example.

  if $SearchMedicines = 0 \wedge$
    $SearchMedicalEquipment = 0 \wedge$
    $VerifyUserLoginInfo = 2 \wedge Products = 0$ then
    tranto ($SearchMedicines = 1 \wedge$
      $SearchMedicalEquipment = 1$) by
      <mean-trig-time, trig-dev>;
  endif;

- *Transition rules for activity completion:* For every basic activity α, a transition rule is further generated, whose conditional statement holds for states where:

  1 α is active
  2 the partner that is used by α is operational.

  Regarding the transition statement of the rule, we have:

  - If α is encapsulated in a while activity, then the transition statement prescribes transitions to target states whose common feature is that α is reactivated. These transitions actually model that in the source states, α executes during the ith iteration of the while activity, whilst in the target states, α executes during the ith + 1 iteration of the while activity.
  - Otherwise, the transition statement specifies transitions to target states, whose common feature is that α is complete.

  The transitions described by the aforementioned rule are characterised by the mean-completion-time (and the completion time deviation) of α.

  Following, an example of a rule is given for the completion of the `BuildCustomerOrder` activity.

  if $BuildCustomerOrder = 1 \wedge Products = 0$ then
    tranto ($BuildCustomerOrder = 2$) by
      <mean-completeion-time, compl-dev>;
  endif;

- *Transition rules for activity failure:* For every basic activity α, two transition rules are specified. The first one is for the generation of transitions that model the failure of activities that already execute, while the second rule is used for the generation of transitions that model the failure of activities that are ready to execute. For the first rule, the conditional statement holds for states where:

  1 α is active
  2 the partner that is used by α is operational.

  The transition statement of the rule states that there should be transitions from the aforementioned states to target states, whose common feature is that the partner is failed. In the target states, α is also considered as

failed if the partner does not represent a redundancy schema. Otherwise, α remains active until the number of schema failures exceeds the value of the $no_{failure}$ property. The rate for these transitions is the failure-rate or the benign-to-active-rate of the partner. An example of such a transition rule was given earlier in Section 4.1.

For the second rule, the conditional statement holds for states where:

1 α is inactive

2 the activities upon which α depends are complete

3 the partner that is going to be used by α is failed.

The transition statement of the rule states that there should be transitions to target states where α is failed.

5 Illustrations

The use and appropriateness of the two reliability analysis techniques proposed in Sections 3 and 4 is illustrated based on the reference example. As discussed in Section 1, the main objective of the pharmaceutical company is to reduce the product delivery time by setting up a local piecemeal delivery service as an alternative to the delivery company that was used until now. Recall that the IT team that took over the reliability assessment of this expansion plan is going to perform a what-if analysis. Briefly, this means to inspect how changes into certain independent parameters impact on certain dependent variables (Golfarelli et al., 2006). Typically, performing what-if analysis involves making some assumptions (scenarios in other terms), based on a brainstorming approach among the IT experts, who rely on historical data mined from the company's organisational memory [the interested reader may further refer to Golfarelli et al. (2006) for a detailed methodology on what-if analysis]. In the particular reference example, the IT team decides to proceed with respect to the following assumptions:

- an early release of the `LocalDelivery` service fails twice as much as the `DeliveryCompany` service that was used until now:

$$failure\text{-}rate_{LocalDelivery} = 2 * failure\text{-}rate_{DeliveryCompany}$$

- a mature release of the `LocalDelivery` service fails as much as the `DeliveryCompany` service:

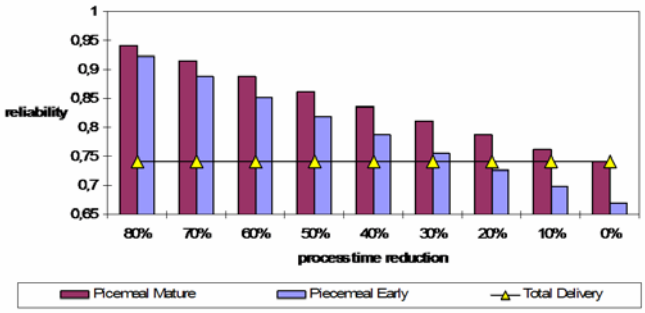
$$failure\text{-}rate_{LocalDelivery} = failure\text{-}rate_{DeliveryCompany}$$

- the execution time of the company's process may be reduced from 0% to 80%, due to the use of the `LocalDelivery` service, depending on how efficient would be the implementation of this service.

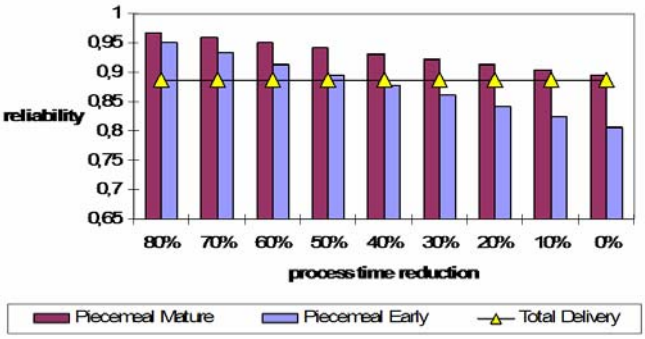
More specifically:

- Based on the proposed methodology, the block diagrams and the Markov models were constructed for the company's process execution path which uses the *early release* of the `LocalDelivery` partner (i.e., the piecemeal delivery path). In these models, the time reduction in the execution time of the company's process (which was initially set to 100 time units) varied from 0% to 80%. The failure rate of the early release of the `LocalDelivery` partner was set to 0.002 failures per time unit, while the failure rates of the rest of the partners involved in this path were set to 0.001 failures per time unit. The reliability values that resulted from these block diagrams and Markov models correspond to the *Piecemeal Early* columns, given in Figures 5(a) and 5(b), respectively.
- Similarly, block diagrams and Markov models were developed for the company's process execution path which uses the *mature release* of the `LocalDelivery` partner. In this case, the failure rate of the LocalDelivery partner and the failure rates of the rest of the partners involved were set to 0.001. As previously, the time reduction in the execution time of the company's process varied from 0% to 80%. The reliability values obtained correspond to the *Piecemeal Mature* columns, given in Figures 5(a) and 5(b), respectively.
- Finally, a block diagram and a Markov model were developed for the company's process execution path that uses the `DeliveryCompany` partner (i.e., the total delivery path). The execution time for the total delivery path was also set to 100 time units. The failure rates of the partners involved in this path were set to 0.001 failures per time unit. The reliability values obtained correspond to the *Total Delivery* lines given in Figures 5(a) and 5(b), respectively [note that lines were used instead of columns to highlight that in this experiment, the execution time of the company's process was not varied; therefore, the lines should not be related with the values that are given in the x-axis of Figures 5(a) and (b)].

Figure 5 Reference example reliability results, (a) block diagrams (b) Markov models (see online version for colours)



(a)



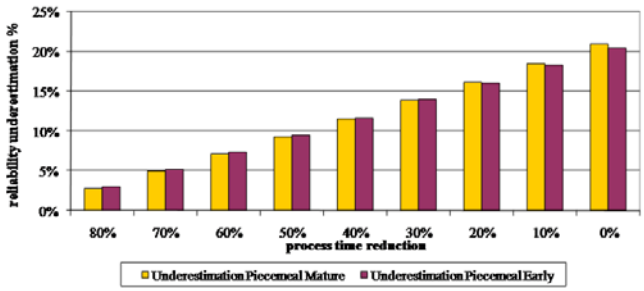
(b)

In all models, the number of items to be delivered was set to ten. Moreover, the partner failures were due to permanent faults and none of the partners represented a redundancy schema. The system's reliability was computed via the block diagrams using a Microsoft Excel spreadsheet. The Markov models were given as input to the SURE tool that took care of all the necessary computations (including the system of first-order equations) and produced the estimation of the system's reliability as an output.

Regarding the results in Figure 5(a), we can observe that the reliability values increased with the efficiency of the `LocalDelivery` partner (both in the Piecemeal Early and the Piecemeal Mature columns). This is reasonable considering that the faster the process gets, the less probable it is to fail during its execution. The total delivery line in Figure 5(a) gives the overall reliability for the process execution path that uses the DeliveryCompany partner. Clearly, this path was less reliable from the process execution path that used the mature release of the `LocalDelivery` partner. The reason behind this is that in both paths, the same failure rates were used for the partners and the execution path that involved the mature release of the `LocalDelivery` partner executed faster. Even the process execution path that used the early release of the `LocalDelivery` partner was more reliable from the path that used the `DeliveryCompany` partner, in certain cases where the authors assumed that the IT team implemented the `LocalDelivery` partner efficiently enough (i.e., in cases where the execution time reduction was greater than 30%).

Figure 5(b) gives the reliability values obtained from the Markov models. The gross observations resulted from the block diagrams still remain valid. However, a comparison between the reliability values obtained from the block diagrams and the Markov models shows that the former underestimated the overall process reliability (Figure 6). This underestimation is due to the fact that in the block diagrams, it is assumed that all partners should be operational for the whole duration of the process. On the contrary, the Markov models allowed to perform a more fine-grained modelling by taking into account the mean-completion times of the activities that constitute the process. The Markov models that were constructed reflected that the partners should be operational only during the activities that use them. The previous increased the reliability values that were calculated.

Figure 6 Block diagrams and Markov comparison (see online version for colours)



Hence, a 'first lesson learned' is that the gross-grained modelling capability of block diagrams is sufficient for driving a reliability analysis of business processes, which may serve for studying reliability trends regarding certain reliability properties (e.g., process execution time). On the other hand, the block diagrams cannot capture dynamics that relate to the execution of the activities that constitute the business processes. Ignoring such dynamics may lead to less precise results. In the reference example, for instance, the block diagrams analysis shows that if the IT group of the pharmaceutical company manages to build an early release of the `LocalDelivery` partner that reduces the process execution time by 30%, then the company's process shall be more reliable despite the fact that the failure rate of the `LocalDelivery` partner is twice as the failure rate of the `DeliveryCompany` partner. According to the Markov models, the corresponding percentage of execution time reduction is much higher (40%). Quite expectedly, a 'second lesson learned' is that Markov models are more suitable for detailed business process reliability analysis. As theoretically anticipated, Markov models are more expressive when facing the issue of modelling system dynamics such as dependent failures and repairable behaviour (Raussand and Hoyland, 2004). Nevertheless, there exist interesting recent works that propose extensions of block diagrams, which overcome the modelling limitations of conventional block diagrams (e.g., Distefano and Puliafito, 2007).

The price to pay for the fine-grained modelling capabilities of Markov models is the complexity of their specification. The models used in the reference example consisted of more than 50 states and more than 100 transitions. The time to calculate reliability values out of the Markov models was comparable to the time required for the block diagrams (less than 20 msec). However, the complexity of the Markov models further increases if we consider partners that fail due to temporary faults [a small example of such a case is given in Zarras et al. (2004)]. In this case, block diagrams are significantly faster than Markov models.

6 Related works

The issue of quality analysis (e.g., performance, reliability, availability, etc.) for conventional composite systems has been explored in the past (Klein et al., 1999; Kazman et al., 2000; Zarras and Issarny, 2000; Zarras et al., 2003; Rodrigues et al., 2003; Majzik et al., 2003; Skene and Emmerich, 2003; Rodrigues et al., 2004; Cardoso et al., 2004). There are both similarities and differences with this line of research. On the common side, these approaches share the methodological approach to the problem (i.e., given a certain input, it is mapped to UML – for ease of modelling – and then the UML model is transformed to a model suitable for a well-known dependability analysis technique). On the other hand, there are prominent differences, specifically tailored for the case of web services: both the input (BPEL in our case) and the systematic derivation of the reliability analysis models are different. The lesson here is that although one does not need to reinvent the wheel in terms of fundamental techniques (but rather, follow a principled methodology), there are still important issues to address, which are handled in this paper.

The OMG (2004) standardisation community recently recognised the importance of modelling the quality of composite systems and adopted a corresponding UML profile. The properties defined in Section 2.2 are aligned with the aforementioned standard. They constitute a superset of the dependability characteristics mentioned in the standard and they are specifically tailored to the case of BPEL business processes.

In the context of web services, the issues of quality specification, analysis and management gained the attention of various research communities. More specifically, in Dan et al. (2004), the authors propose a framework for the provision of differentiated levels of service that meet the customers' functional and quality requirements, which are described in terms of service level agreements (SLAs). SLAs are specified using a declarative language, named WSLA. SLAng is also a language for the specification of SLAs (Skene et al., 2004). While these approaches are quite generic, the proposed approach focused on reliability properties and reliability analysis techniques for service-oriented business processes. The reliability properties defined can be seen as SLA attributes. Then, the models can be used to further generate WSLA or SLAng specifications. In Cardellini et al. (2001), an infrastructure-based solution is proposed for the provision of differentiated levels of service. It particularly deals with performance SLA attributes. Similarly, in Liu and Issarny (2003), the problem of locating basic web services in ad-hoc networks based on a set of quality criteria is tackled. In Zeng et al. (2003), a similar problem has been dealt with. More specifically, in this approach, the input is the specification of a process that combines N primitive web services. Moreover, the authors assume the existence of N sets of compatible primitive services characterised by a number of quality attributes like reliability, performance, price, reputation, etc. Then, they propose a technique that allows selecting N services out of the N sets, which provide optimal process quality. Although the proposed approach is interesting, its reliability analysis part can be refined based on the methods proposed in this paper.

7 Conclusions

This paper, investigated methods that enable the what-if reliability analysis of business processes. Specifically, a UML method for modelling business processes was proposed. The proposed method is built upon BPEL and introduces necessary extensions to support the specification of reliability properties that characterise the constituents of business processes. Moreover, systematic methods were introduced for using the resulting BPEL-specific UML models as input to two well-known reliability analysis techniques that rely on block diagrams and Markov models, respectively. Finally, the use of these techniques was illustrated and their appropriateness regarding their precision and the resources they require was discussed.

The formal foundations for the correctness and reversibility of the mapping from BPEL to UML have not been dealt with in this paper and constitute an interesting research topic. Moreover, in this paper, it is assumed that the end-users of the proposed approach are the designers of web service based information systems, who statically perform the analysis at design-time. Performing reliability analysis dynamically during the lifetime of the system is also an interesting research issue. Such functionality can be part of a middleware infrastructure that supports the development of composite web services. In the latter case, the complexity of the analysis techniques plays an even more important role, especially if the infrastructure is targeted to the development of web services in pervasive computing environments (Issarny et al., 2005).

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments. This work was partially funded by the MobWS GSRT grant for Cooperation in S&T areas with European countries.

References

Butler, R.W. (1992) 'The SURE approach to reliability analysis', *IEEE Transactions on Reliability*, Vol. 41, No. 2, pp.210–218.

Cardellini, V., Casalicchio, E., Colajanni, M. and Mambelli, M. (2001) 'Web switch support for differentiated services', *ACM SIGMETRICS Performance Evaluation Review*, Vol. 29, No. 2, pp.14–19.

Cardoso, J., Sheth, A., Miller, J., Arnold, J. and Kochut, K. (2004) 'Quality of service for workflows and web service processes', *Journal of Web Semantics*, Vol. 1, pp.281–308.

Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M. and Youssef, A. (2004) 'Web services on demand: WSLA-driven automated management', *IBM Systems Journal*, Vol. 43, No. 1, pp.136–158.

Distefano, S. and Puliafito, A. (2007) 'Dynamic reliability block diagrams vs. dynamic fault trees', in *Proceedings of the IEEE Reliability and Maintainability Symposium (RAMS'2007)*, pp.71–76.

Eckhardt, D.E. and Lee, L.D. (1985) 'A theoretical basis for the analysis of multiversion software subject to coincident errors', *IEEE Transactions on Software Engineering*, Vol. 11, No. 12, pp.1511–1517.

Golfarelli, M., Rizzi, S. and Proli, A. (2006) 'Designing what-if analysis: towards a methodology', in *Proceedings of the 9th ACM International Workshop on Data Warehousing and OLAP*, pp.51–58.

IBM, Microsoft Corporation and BEA (2002) *Business Process Execution Language for Web Service (BPEL4WS) v.1.0*, Technical Report, IBM, Microsoft Corporation, BEA, available at http://www.ibm.com/developerworks/webservices/library/ws-bpel/.

Issarny, V., Kloukinas, C. and Zarras, A. (2002) 'Systematic aid for developing middleware architectures', *Communications of the ACM (CACM)*, Vol. 45, No. 6, pp.53–58.

Issarny, V., Sacchetti, D., Tartanoglou, F., Sailhan, F., Chibout, R., Levy, N. and Talamona, A. (2005) 'Developing ambient intelligence systems: a solution based on web services', *Journal of Automated Software Engineering*, Vol. 12, No. 1, pp.101–137.

Johnson, S.C. (1988) 'Reliability analysis of large complex systems using ASSIST', in *Proceedings of the 8th AIAA/IEEE Digital Avionics Systems Conference*, pp.227–234.

Johnson, S.C. and Boerschlein, D.P. (2000) *ASSIST User Manual*, January, NASA Langley Research Center.

Kazman, R., Carriere, S.J. and Woods, S.G. (2000) 'Toward a discipline of scenario-based architectural engineering', *Annals of Software Engineering*, Vol. 9, pp.5–33.

Klein, M., Kazman, R., Bass, L., Carriere, S.J., Barbacci, M. and Lipson, H. (1999) 'Attribute-based architectural styles', in *Proceedings of the 1st IFIP Working Conference on Software Architecture (WICSA-1)*, pp.225–243.

Knight, J.C. and Leveson, N.G. (1986) 'An experimental evaluation of the assumption of independence in multi-version programming', *IEEE Transactions on Software Engineering*, Vol. 12, No. 1, pp.96–109.

Laprie, J-C. (1985) 'Dependable computing and fault tolerance: concepts and terminology', in *Proceedings of the 15th International Symposium on Fault-Tolerant Computing (FTCS-15)*.

Laprie, J-C., Arlat, J., Beounes, C. and Kanoun, K. (1990) 'Definition and analysis of hardware and software fault-tolerant architectures', *IEEE Computer*, Vol. 23, No. 7, pp.39–51.

Liu, J. and Issarny, V. (2003) 'QoS-aware service location in mobile ad-hoc networks', in *Proceedings of the 5th IEEE International Conference on Mobile Data Management (MDM'04)*.

Majzik, I., Pataricza, A. and Bondavalli, A. (2003) *Architecting Dependable Systems, LNCS, Chapter Stochastic Dependability Analysis of System Architecture Based on UML Models*, Vol. 2677, pp.219–244, Springer-Verlag.

Mantell, K. (2003) *From UML to BPEL*, Technical Report, IBM, available at http://www.106.ibm.com/developerworks/webservices/library/ws-uml2bpel/.

OMG (2004) *UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms*, Technical Report, OMG, ptc/2004-06-01, available at http://www.omg.org/docs/ptc/04-06-01.pdf.

Rausand, M. and Hoyland, A. (2004) *System Reliability Theory Models Statistical Methods and Applications*, 2nd ed., Wiley.

Rodrigues, G., Rosenblum, D. and Emmerich, W. (2004) 'A model driven approach for software systems reliability', in *Proceedings of the 26th IEEE/ACM/SIGSOFT International Conference on Software Engineering (ICSE'04)*, pp.30–32.

Rodrigues, G.N., Roberts, G., Emmerich, W. and Skene, J. (2003) 'Reliability support for the model driven architecture', in *Proceedings of the 2nd IEEE-ACM-SIGSOFT ICSE Workshop on Software Architectures for Dependable Systems (WADS'03)*, p.7.

Skene, J. and Emmerich, W. (2003) 'A model driven architecture approach to analysis of non-functional properties of software architectures', in *Proceedings of the 18th IEEE Conference on Automated Software Engineering (ASE'03)*, pp.236–239.

Skene, J., Lamanna, D. and Emmerich, W. (2004) 'Precise service level agreements', in *Proceedings of the 26th IEEE/ACM/SIGSOFT International Conference on Software Engineering (ICSE'04)*, pp.179–188.

W3C (2001) *Web Services Description Language (WSDL) v1.1*, Technical Report, W3C, available at http://www.w3c.org/TR/wsdl.

W3C (2002) *Simple Object Access Protocol (SOAP) v1.2*, Technical Report, W3C, available at http://www.w3c.org/TR/soap12-part0.

Zarras, A. and Issarny, V. (2000) 'Automating the performance and reliability analysis of enterprise information systems', in *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01)*, pp.350–355.

Zarras, A., Kloukinas, C. and Issarny, V. (2003) *Architecting Dependable Systems, LNCS, Chapter Quality Analysis of Dependable Systems: A Developer Oriented Approach*, Vol. 2677, pp.197–218, Springer-Verlag.

Zarras, A., Vassiliadis, P. and Issarny, V. (2004) 'Model-driven dependability analysis of web services', in Meersman, R., Tari, Z. et al. (Eds.): *Proceedings of the 6th International Symposium on Distributed Objects and Applications (DOA'2004)*, LNCS, No. 3290, pp.1608–1626.

Zeng, L., Benatallah, B. and Dumas, M. (2003) 'Quality driven web services composition', in *Proceedings of the 12th ACM International Conference on the World Wide Web (WWW'03)*, p.411.