# Fault Tolerance Framework using Model-Based Diagnosis: Towards Dependable Business Processes

Angel Jesus Varela-Vaca, Rafael M. Gasca, Diana Borrego, Sergio Pozo
*Computer Languages and Systems Department,*
*Quivir Research Group*
*ETS. Ingeniería Informática, Avd. Reina Mercedes S/N,*
*University of Seville, Seville, Spain*
{*ajvarela, gasca, dianabn, sergiopozo*}*@us.es*

***Abstract*—Several reports indicate that one of the most important business priorities is the improvement of business and IT management. Management and automation of business processes have become essential tasks within IT organizations. Nowadays, business processes of a organization use external services which are not under our its jurisdiction, and any fault within these processes remain uncontrolled, thereby introducing unexpected faults in execution. Organizations must ensure that their business processes are as dependable as possible before they are automated. Fault tolerance techniques provide certain mechanisms to decrease the risk of possible faults in systems. In this paper, a framework for developing business processes with fault tolerance capabilities is provided. Our framework presents various solutions within the scope of fault tolerance, whereby a practical example has been developed and the results obtained have been compared and discussed. The implemented framework presents innovative mechanisms, based on model-based diagnosis and constraint programming which automate the isolation and identification of faulty components, but it also includes business rules to check the correctness of various parameters obtained in the business process.**

***Keywords-Business process, Business Process Management, Fault-tolerance, Dependability.***

## I. INTRODUCTION

The automation of business processes is emerging into the enterprise arena as a mechanism for improvement. Companies may decide to deploy Business Process Management System (BPMS) to automate their business processes, but it system cannot guarantee perfect executions error-free or fault-free execution. On the other hand, nowadays there exist a trending in the integration of services of different companies in the business processes. The integration of external services set up new point of vulnerability in the business process execution. Companies have to ensure that their business processes are as dependable as possible using mechanisms such as introduced in [1][2]. Gartner's CIO report [3] indicates that the most important business priorities include: improvement of business processes, cost reduction, enterprise workforce effectiveness, security and IT management. Therefore, dependability is a significant requirement for many types of companies, since any failure in their business processes may lead to terrible consequences: economic lost, lives lost, systems destroyed, security breaches, and so on.

In recent years, a new paradigm has emerged in the scope of business IT: Business Process Management (BPM). BPM is defined as a set of concepts, methods and techniques to support the modelling, design, administration, configuration, enactment and analysis of business processes [4]. BPM has become an essential tool for organizations, since it is defined as a methodology for the improvement of the efficiency through systematic management of business processes that should be modelled, automated, integrated, monitored and optimized in a continuous way. One of the most important goals of BPM is the better understanding of the operations that a company performs and the relationships between these operations. BPM also aims at narrowing the gap between business processes that a company performs and the implementation of these processes in the BPMS.

The BPM paradigm follows a life cycle that consists of several stages [5], shown in Figure 1. During each stage, various kinds of faults can be introduced:

- In the design stage, business process models can present some design faults (such as deadlocks, live-locks and starvations). Some systematic approaches provide design guidelines that allow their designed processes to be corrected and improved. Design problems are not taken into account in this paper since it is an issue that has already been subject to wide discussion [6][7][8].
- In the run-time stage, faults could be located in the business processes when unexpected outputs, unexpected messages, unexpected events, or unexpected performances are obtained. Executable business processes tend to use external services that are not under their jurisdiction. Thus, it is impossible to ensure that the functionality of a certain external service changes during the business process life cycle. As a consequence, unexpected changes in services could entail changes in business process behaviour.

Therefore, companies must pay attention on the inclusion

of measures that promote the reduction of the risk of possible faults and increase the dependability of business processes from design stages. The majority of proposals have used fault tolerance ideas in other areas such as: firewalls, grid computing, composition of applications and service-oriented architectures, [9][10][11][12][13]. In these studies, various fault tolerance approaches have been applied: check-point view [9], replication and recovery techniques [10][11], and other sophisticated techniques such as dynamic binding [12], and self-reconfiguration of systems [13]. Our approach proposes to improve dependability properties in the execution of business processes based on techniques for automatic identification of faults by means of model-based diagnosis which helps to establish specific fault tolerance mechanisms. Fault tolerant mechanisms applied in this work are based on classic fault tolerance ideas such as replication, check-pointing and techniques of diversity focused on the context of service-oriented business processes.

This paper is structured as follows: Section II introduces some concepts of BPM and fault tolerance; Section III presents our framework for dependable business processes using fault-tolerant techniques; in Section IV, a practical example is explained and developed; Section V shows experimental results that are discussed; and, in the last section, conclusions are drawn and future work is proposed.

## II. Business Process Management System and Fault Tolerance

In order to understand the BPM paradigm, it is necessary to show the typical business process life cycle [5], as shown in Figure 1. The life cycle consists of different stages:

1) **Design and Analysis:** business process models are defined and validated.
2) **Configuration:** once business process models are validated, they need to be implemented by a dedicated software system. In this configuration stage a software systems (BPMS) is chosen and configured where business processes will be deployed.
3) **Enactment:** once deployed, process enactment needs to guarantee correct execution in accordance with various constraints specified in the model.
4) **Diagnosis:** techniques are applied to identify and isolate faults in business processes. Nowadays, most diagnosis techniques applied are focused on identifying design faults.

In BPM, several levels of business processes can be identified depending on the point of view of the organization [4]. In this paper, only two levels of BPM are considered:

- Operational business processes are described with business process models where the activities and their relationships are specified, but implementation aspects are not taken into account. Operational processes are the basis for developing implemented business processes.

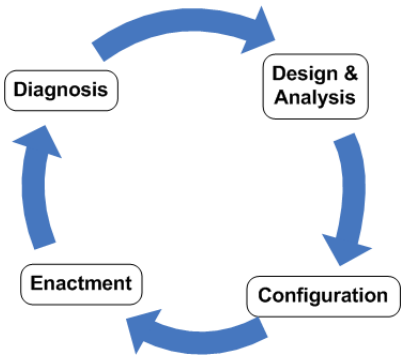



Figure 1. Business process life cycle.

- Implemented business processes contain information about the execution of the activities and the technical and organizational environment where they will be deployed and executed.

Implemented business processes and specifically service-based business process use external services which remain outside the jurisdiction of the organization. Although, business process models are validated and verified at the design stage, organizations cannot guarantee the correct execution of changes in services interms of functionality and faults simply through the lack of response or security attacks, and so on. The identification of where business processes are failing, and which components are involved in the faults may probe worthwhile for the numerous business stakeholders (designers, analysts, developers, etc).

Fault diagnosis is a method which permits us to determine why a correctly designed business process fail to work as expected. Diagnosis aims to identify and isolate the reason of any unexpected behaviour, or in other words, to identify which parts are failing in a business process. In this work, model-based diagnosis [14] is used in order to isolate the faulty services. Model-based diagnosis is recognized as a very powerful tool within the community of diagnosis due to its ability to solve the problem of isolating faulty components.

Our proposal is focused on achieving dependable business processes, but to the achievement of dependability must first be clarified [15]. The properties for achieving dependability fall into four major groups: fault tolerance, fault avoidance or prevention, fault removal, and fault forecasting.

By definition, fault tolerance [15] is a mechanism used to order to guarantee service by complying with the specification in spite of the presence of faults. Fault-tolerance techniques are a means of reducing the risk of faults. On the whole, fault-tolerance frameworks are focused on physical systems and not on software systems and most applied techniques are based on replication and recovery. Replication is employed in the recuperation of services by means of duplication of each of its functionalities in form of replicas

and in the case of a service replica fault another replica takes control. Typical solutions in replication are considered as:

- *Passive replication* [16]: the client only interacts with one replica (primary) which handles the client request and sends back responses. The primary replica also issues messages to the backup replicas (other secondary replicas) in order to update their state.
- *Active replication* [16]: all replicas play the same role. All replicas receive each request, handle the request, and send back the response to the client. Other solutions based on active replication [17] exist.

Initially, active replication provides a generally faster response time than passive replication. However, in active replication all replicas must to process the requests, and hence more system resources are employed than for passive replication. Moreover, the nested redundant replicas could cause a problem of invocations since the use of active replication requires replicas to be deterministic. No such determinism is required for passive replication, and therefore it is more flexible.

In the majority of cases, fault tolerance is used as only a hardware approach, but software is also a crucial factor in the effective good running of organizations. Increasing the dependability of software presents some unique challenges when compared to increasing the dependability of traditional hardware systems [18]. Hardware faults are mainly physical fault, which can be characterized and predicted over time. Software has only logical faults which are difficult to visualize, classify, detect, and correct. Changes in operational usage or incorrect modifications may introduce new faults. To protect against these faults, it is insufficient to simply add redundancy, as is typically done for hardware faults, since doing so would simply duplicate the problem.

## III. FAULT TOLERANCE FRAMEWORK

In the previous section, the basic ideas of the business process life cycle are introduced and the main stages to manage business processes are shown. This section is focused on presenting the framework by defining all its components and clarifying how it works. The proposed framework is based on the main ideas of the BPM life cycle, whose structure is depicted in Figure 2. As can be observed, the framework is structured in four main layers: Modelling, Applications, Fault Tolerance, and Services. In the following subsections, the various parts of the framework are detailed and discussed. Although it can be noticed that the main characteristic is the utilization of a specific fault-tolerance layer. This layer contains specific mechanisms in order to mitigate the risk of possible process faults detected in the execution of business processes.

Before continuing any further, some assumptions about business processes should be stated:

1) A business process model has a single start event, a single end condition and every activity contributes towards finishing the process correctly.
2) The business process design is correct, and hence no design faults exist(deadlocks, live locks, starvations, and so on).
3) Operational business processes cannot suffer byzantine faults (arbitrary faults).
4) Operational business processes are stateless.

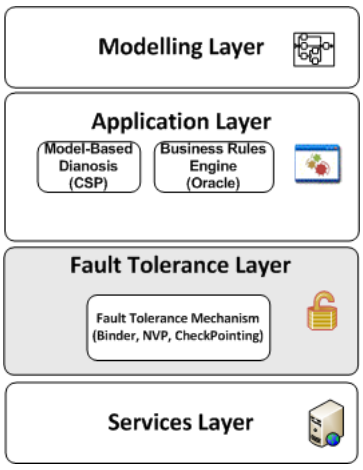



Figure 2. General view of the framework.

Therefore, the main problems left for discussion in this paper involve the diagnosis and mitigation of incorrect outputs/results and events in business processes. In the following subsections, each framework layer is described.

### A. Modelling Layer

The design stage is focused on describing different models used in our framework. Our approach is based on three kinds of models:

- Business process models are graphical descriptions of business processes (BP). These models represent the graphical formalization of the various constraints to which a business process has to comply at any time they are employed. Within the arena of BPM, various modelling languages have emerged: Flowcharts, Petri Nets, Event-driven Process Chain (EPC), UML Activity Diagrams, Data Flow Diagrams (DFD), IDEF, and Business Process Management Notation (BPMN). In the design stage, the most widely used for business processes representation is currently the standard Business Process Management Notation (BPMN) by OMG [19]. This notation is highly useful for business analysts since BPMN contains many different elements such as activities, data objects, and gateways. However, BPMN diagrams save no information on where they will be executed and are platform-independent. Figure 3 shows an example of a BPMN diagram where elements are described. Our framework can support BPMN diagrams,

and a prototype is developed and shown in [6].

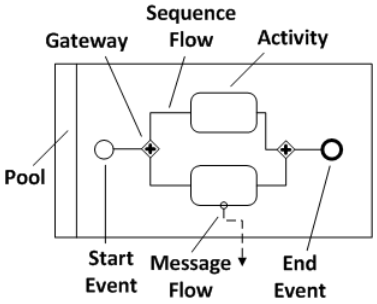



Figure 3. BPMN element description.

- Business rule models; a business rule is a statement that defines those aspects of business processes that are impossible to gather or express in the model [20]. In fact, by separating business logic from business design, if the business process logic changes then, only the business rules must be changed but not the business process. Business rules can be formalized as ”if-then” statements in a selected Business Rule Management Systems (BRMS), by using natural language or formal methods. BRMS is a software system employed to define, deploy, execute, monitor, and maintain the variety and complexity of decision logic that is used by operational systems within an organization. In our proposition, business rules act as an ”oracle”. Thus, they indicate the correctness of outputs of the business processes.
- Constraint models are necessary in the model-based diagnosis stage where Constraint Satisfaction Problems (CSP) are used. In our proposition, constraint models are employed to describe behaviour activities gathered within business processes. Constraint models can be constructed off-line or on-the-fly. Process constraint models can be built online since logs captured where information of inputs, outputs and service constraint model can be given. Both Diagnosis and CSPs will be described in Section III-C.

Business process validation analysis methods are focused on discovering whether the designed business processes can be automatically enacted as expected. Through this analysis, any possible constraint violations should be detected. In various studies, process models have been analyzed in order to prevent structural faults in the form of : Petri Nets [21]; a set of graph reduction rules to identify structural conflicts in process models [22]; and an improved version of the latter method [23]. However, in a earlier work [6] a business process validation has been studied, and an automatic mechanism to fault structural diagnosis has been integrated in a editor of BPMN diagrams.

### B. Application Layer

In this layer, various technologies which are used to implement and deploy models are introduced. Business Process Execution Language (BPEL) is a de facto standard language for the implementation of service-based business processes. The main advantages in using BPEL as an implementation language are:

- BPEL supports all necessary elements in the implementation of sophisticated business processes, such as those in BPMN, and provides a sufficient number of mechanisms to implement fault tolerant mechanisms [24].
- BPEL processes are specific implementations of business processes for service-oriented environments, as suggested in [4].
- The majority of the distinguished commercial and non-commercial tools support service-oriented architectures and BPEL processes as shown in Table I.

Table I
COMPARATIVE OF BPM TOOLS.

| | Process Modeling | Methodology | SOA | Support BPEL | Translation BP2BPEL |
|---|---|---|---|---|---|
| Intalio BPMS | √ | BPMN | √ | √ | √ |
| IBM WepSphere Process Integration | √ | No Standard | √ | √ | √ |
| Appian BPM Suite | √ | BPMN | √ | - | - |
| Tibco iProcess Suite | √ | BPMN | x | x | x |
| PegaSystem SmartBPM Suite | √ | - | - | - | - |
| Oracle BPM Suite | √ | BPMN | √ | √ | ¿ |
| G360 Enterprise BPM Suite | √ | BPMN | x | x | x |
| Lombardi BluePrint & Teamwork | √ | BPMN | x | x | x |
| Savvion BusinessManager Platform | √ | BPMN | √ | √ | x |
| Fujitsu Interstage BPM Suite | √ | BPMN | √ | √ | x |
| IBM FileNet BP Manager | √ | BPMN | √ | x | x |
| Aura Portal BPMS | √ | BPMN | √ | x | x |

Although BPMN is a standard notation for the design of process models, it could automatically be translated into BPEL [19][25][26], and most commercial BPMSs support BPMN and BPEL processes. Therefore, BPEL processes represent the best candidate and will be used in our proposal. A BPEL environment is necessary for the implementation and deployment operational business process. In this proposal, GlassFishESB have been integrated in our framework. NetBeans provides an environment in order to develop BPEL

processes graphically. Moreover, GlassFishESB provides support as an application server with several components of Enterprise Service Bus (ESB) [27] and a runtime for BPEL processes.

There is no accepted standard for business rule systems. For this reason, in our proposal business rules have been developed using WebSphere Ilog JRules and have been integrated within business processes as services.

Neither is there any standard CSP representation nor CSP Solver. For the development of constraint models, any CSP Solver could be used. In our case, constraint models have been implemented using ChocoSolver [28] which has been integrated into our framework. However, the model could be implemented as a standalone from other CSP Solver tools, and used as a service.

### C. Diagnosis

In our approach, model-based diagnosis is applied. Diagnosis is used for the identification and isolation of behavioral faults in the components of business processes. Constraint Satisfaction Problem (CSP) techniques have been applied in the diagnosis since CSP is an extended technique which solves a wide variety of problems, including those in business processes [29]. In order to apply CSP, business processes have been transformed into Constraint Optimization Problems (COP) [30]. COPs are specific CSPs where an objective function to optimize remains to be optimized. COP is evaluated using a CSP solver (a solver is an engine of constraints that implements algorithms in order to solve constraint satisfaction problems). The diagnosis will only be invoked when a fault in the outputs of the business processes has been detected.

### D. Fault Tolerance Layer

Framework has been built with a specific fault tolerance layer. This layer is developed with the intention of controlling possible faults and taking corresponding corrective actions in order to recover the execution business processes, and then to achieve dependable properties. In our case, some replication solutions have been adopted, but on the other hand, software fault tolerance techniques have also been considered. Likewise, replication and recovery techniques are focused on the redundancy of components without considering the business process state. Therefore, one fault tolerance mechanism is focused on the simulation of checkpoint and recovery [31], but in this case oriented towards business processes. Various fault-tolerant mechanisms are described below:

1) **Without fault tolerance**
   BPEL processes are defined as a composition of web services. Thus, web services (of either external or internal organization) are linked at design time. If a fault output or event occurs (supposing no design faults exist in the processes), this fault is located on the service side, for instance, a change of functionality, or a miss-match of parameters. Once the BPEL process has been deployed, it is impossible to replace the faulty service during run-time. Faults are only solved by stopping for every instance of a business process. Firstly, the faulty service has to be located (diagnosis), then eliminated, and finally replaced with another correct service. Therefore, it is necessary to introduce mechanisms to mitigate effects produced by faults without having to stop the execution of business processes and this can be carried out by means of fault tolerance techniques.
2) **Primary/Backup approaches**
   This solution applies the concept of redundancy, in the sense of replication of services. This approximation has a primary service as principal and one or more replicas as backups. In the case of a fault, it is possible to use the backup services. The adopted solution in the proposal is based on dynamic binding ideas [32][33]. Dynamic binding is a technique that allows services to be linked at run-time.
   In our approach, a binder component is introduced between the BPEL processes and the services acting as proxies, as shown in Figure 4. Binder is not developed as an external program or external monitor, but as another business process. The binder component decides at run-time what services to invoke: whether they be the original service or the replicas (backups). The solution is fault tolerant since in the case of detecting a faulty service, every faulty service invocation can be replaced with an invocation to a backup service.

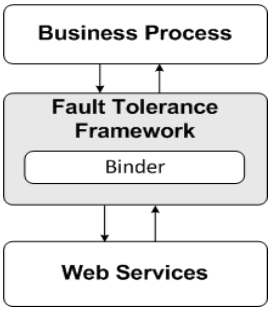



Figure 4. Communication process-service with binder.

This is a good solution despite presenting some deficiencies such as the introduction of a unique point of fault at binder component which in turn could introduce a very high overhead in the performance of a business process. In order to solve the first problem, the replication of the binder can be applied, thereby also, replicating the binder component (one primary and several backup replicas). In our case, passive replication is used in the proposed solution. In the case of a fault, in the first binder, a backup binder

takes the control of the execution, thereby enabling the execution to continue, see Figure 5. The binder backup can be an exact copy of the original but located in another external server or machine that of the primary copy. BPEL provides various mechanisms, such as fault and compensation handlers, to achieve the implementation of primary-backup binders.

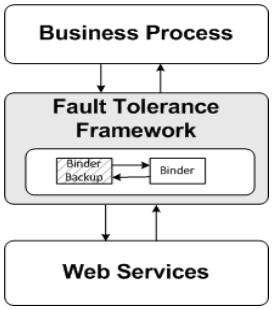



Figure 5. Communication process-service with binder replicated.

3) **Multi-Version (N-Version Programming) approach**
Software components cannot be degraded in the same way as physical systems and redundancy fails to provide a good solution, since if a fault is detected in a software component, it is due to an implementation fault with a very high probability, and this kind of fault cannot be solved by means of redundancy. Diversity is a very important factor in obtaining dependable software systems [18]. The main goal of diversity is to provide identical services (variants) but with a separate design and implementation in order to minimize causing identical faults. For instance, when a software variant presents a fault, this will be isolated as much as possible. There are different techniques for fault-tolerance software based on multi-version (diversity of software) [18][34]: N-Version Programming (NVP), Recovery Blocks, N-Self Checking Programming.
In the proposed framework, a solution based on NVP is adopted, which is a static technique where a activity is executed by various processes or programs and the result is only accepted by majority of votes. This mechanism for obtaining results is defined in NVP as an adjudicator or decision mechanism (DM). Various decision mechanisms are defined in [18].
The N-Version paradigm considers the utilization of diversity for implementations and designs in order to isolate faults in the components. Every service used in the implementation will be developed as an N-Version Component. The implementation selected for N-Version components follow the basic ideas of N-Version Programming. An N-Version component provides at most $2X + 1$ replicas, $X$ ranges from $1$ to N, and where N is an integer greater than 1. Components have been developed with an adjudicator (DM) in order to obtain the output results, see Figure 6. The logic within the adjudicator (DM) can be very basic or seriously complicated. However, NVP components can be improved by adding new features, for example, by developing new strategies in the adjudicator. However, the more complexity added into the adjudicator component, the more overhead is introduced into the execution of the N-Version components.

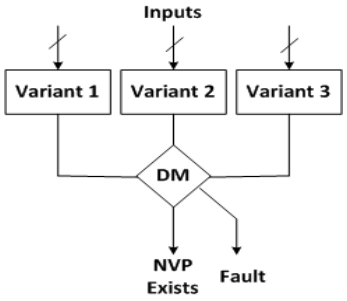



Figure 6. Example of N-Version component.

By using N-Version components, not only is fault-tolerance achieved, but it also supposes another advantage since the diagnosis stage is rendered totally unnecessary. For example, if one of the variants fails return a response in time, the adjudicator takes the results from the other variants. In consequence, diagnosis can be eliminated from the framework by using this mechanism, although it could result in a very high cost in developing and performance.

4) **Checkpointing approach**
The checkpoint mechanism is based on the idea of saving the state of the system, and, in the case of fault detection, recovering the execution of the system from the checkpoint where the state was saved. We propose the simulation of a checkpoint approach in services, whereby a recovery mechanism is launched only in the case of faults. The fault tolerance approach mechanism is composed of two parts:
- Sensors (Checkpoints). An integrity sensor is modelled as a CSP. Sensors receive data information about data inputs and outputs from the services, with which the CSP is then defined. CSP resolutions help to identify and isolate the services which are failing in run-time.
- Compensation handlers (Rollback). These are specific elements of business processes which allow the limitation of the effects created by a process when faults or errors occur. Compensation handlers allow the process execution to be rolled back from a specific point, thereby executing a set of tasks to undo the transactions already initiated. Compensation handlers are explained in next section.

The checkpoint approach presents some drawbacks in comparison with the other approaches: it requires the introduction of extra elements (sensors) into the business process design, extra time to check each sensor, and recovery of business process services in rollback. In fact the correct and minimal localization of sensors inside a business process could be a very highly complex task [35].

For high dependability, the checkpointing solution is not suitable since for long business processes, the rollback mechanism could introduce a very high overhead in the case of a fault. However, if very high dependability in our business processes is needed, then a solution with binder backup is the best solution. Nevertheless, if very high dependability is not needed, then a binder approach alone could be sufficient. If the business process with a very high level of correctness in outputs is needed, then NVP is the best solution. Although an evaluation of the number of replicas needed must be carried out since with a specific number of replicas a very high level of correctness could be ensured despite of the introduction of very high load on the development and in the adjudicator logic.

Table II provides a summarization of some characteristics of fault tolerance mechanisms applied:

- Type. The kind of fault tolerance used. For example, the replication of services in the case of a binder.
- Model-Based Diagnosis. Whether the diagnosis is necessary or not for this technique. For example, in NVP solution the diagnosis stage is unnecessary.
- Overhead. It indicates the additional logic introduced for the development of this technique. For example, in the case of binder, a dynamic binding additional logic is necessary.

## IV. ILLUSTRATIVE EXAMPLE OF APPLICATION

In order to clarify the various alternatives, an example is developed. Although it is a small example, it can perfectly illustrate the problematic under consideration. In this process, there is a set of services, $S = \{S1, S2, S3, S4, S5, M1, M2, M3\}$, a set of inputs, $I = \{a, b, c, d, e, f\}$, and a set of outputs, $O = \{g, h, i\}$. The system is made up of three services *(M1, M2, M3)* with the same functionality *M(x)* but they are independent; and *(S1, S2, S3, S4, S5)* are another five elements with the same functionality *S(x)* but also independent, see Figure 7.

To illustrate a real example, the process has been distributed. The global process has been divided into three separated processes which are deployed in three different systems. The new distributed process is shown in Figure 8. In this case, there is a global process; *"Complete Process"*,

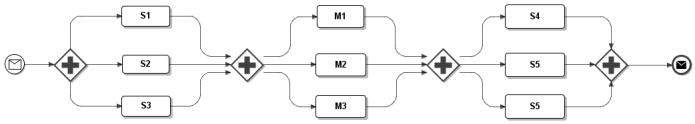



Figure 7. Example of business process.

which orchestrates the other three BPEL processes. BPEL Process 1 contains the invocations to the services S1, S2 and S3. BPEL Process 2 contains the invocations to the services M1, M2 and M3 and relies on BPEL Process 1. BPEL Process 3 contains the invocation to the services S4 and S5 and relies on BPEL Process 2 and BPEL Process 1.

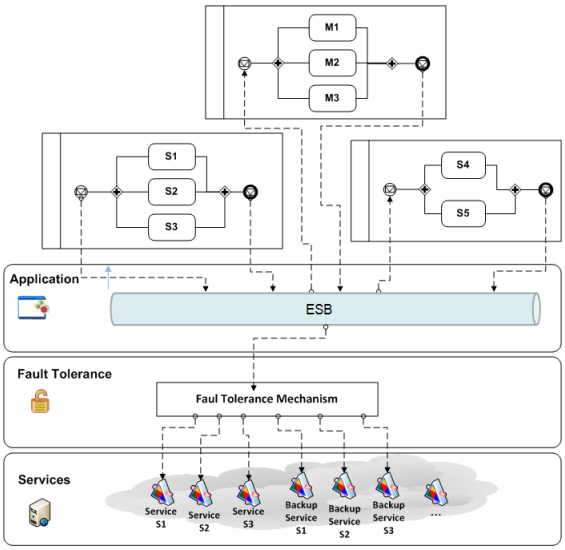



Figure 8. Distributed process.

In the scenario, it is possible to observe numerous aspects. For instance, a fault within service *S1* has a direct effect on the BPEL Process 1 result, and as a consequence the BPEL Process 2 is also affected, and finally BPEL Process 3 will be affected, and hence the final result of the complete process will be not correct. However with the correct fault tolerance mechanism this will not be the case. Therefore, fault tolerance mechanisms are necessary in order to improve the fault tolerance in the execution a business process.

1) **Primary-Backup Services with unique binder approach.** This solution combines the concept of redundant services with dynamic binding mechanisms. This approximation has a primary service as principal and one or more backups. In the case of a fault diagnosis of a service, it will be possible to replace the invocations from this primary with the backup service. A binder component is introduced between the process and the services. In Figure 9 only some services (S1, S2, and S3) have been represented, but every service has to be invoked for the binder component. The

Table II
COMPARATIVE OF THE FAULT TOLERANCE TECHNIQUES.

| | Type | Model-Based Diagnosis | Overhead | Comment |
|---|---|---|---|---|
| **Without Fault Tolerance** | *NA* | + | Cost repair and detection | - |
| **Binder** | Replication (Primary/Backup) | + | Dynamic binding | Binder component |
| **Redundant binder** | Replication (Primary/Backup) | + | Dynamic binding | Replication of binder on external machine of primary |
| **N-Version Programming** | Diversity (N-Version Component) | - | Adjudicator (Voting system) & N-Replicas | Developing N-Version Components |
| **Checkpointing** | Replication (Primary/Backup) | + | Sensors & RollBack Mechanism | RollBack simulation using Compensation Handlers |

binder component is common to every BPEL process. The solution is fault-tolerant although it introduces a drawback since a unique point of fault is located in the binder component. Thus, if a binder fails then all invocations to services will not be produced and all business process executions will not be able to finish.

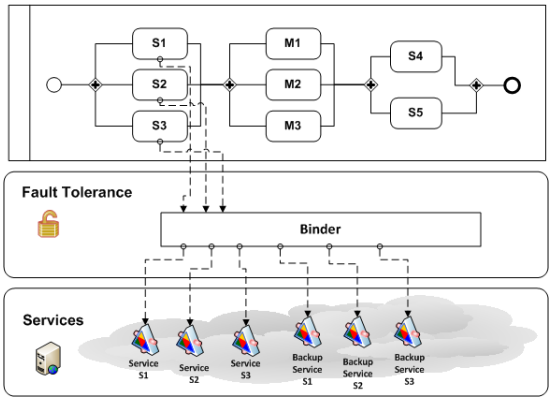



Figure 9. Fault-Tolerant solution with binder.

2) **Primary-Backup Services with replicated binder approach.** In this case a replication of the binder is introduced. If the binder component enters a fault state, then backup replica can take control of the execution, see Figure 10. In the developed proposal, the binder backup is an exact copy of the original. The binder component is common to every BPEL process.
3) **N-Version components approach.** The N-Version components use three variants with an adjudicator to obtain the output results (see Figure 11). The logic within each adjudicator is basic so that no introduce high overhead into the final performance of the process. The adjudicator therefore only compares the outputs from the variants by means of a voting system. In this implementation, a fault-tolerant software component is achieved and the diagnosis stage is rendered totally redundant.

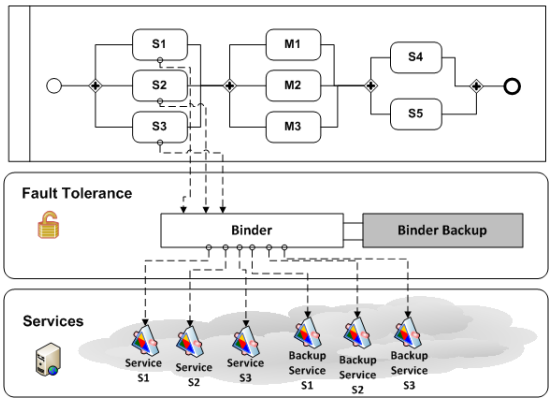



Figure 10. Fault-Tolerant solution with replicated binder.

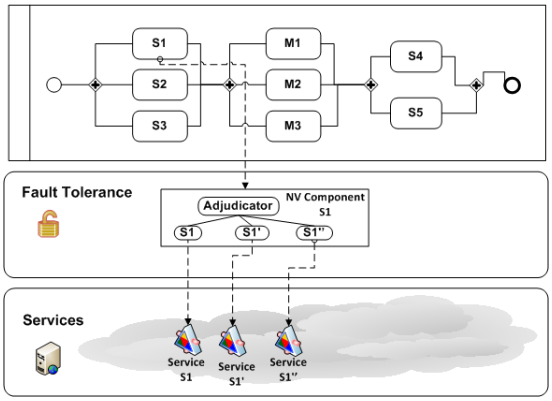



Figure 11. Fault-Tolerant solution with N-Version components.

4) **Checkpointing approach.** It is supposed that the sensors have been correctly located. Sensors, by means of a CSP resolution, indicate whether a service is

behaving as expected. In a checkpointing approach, the process state is saved at this checkpoint, but in our approach this is not necessary. Likewise, sensors provide certain information in order to help the model-based diagnosis stage determine a fault in a finite set of activities of the business process. In Figure 12, sensor IS1 covers the services S1,S2 and S3, while IS2 covers the services from IS1 and IS2.

In order to explain the functionality of a compensation handler, the following example is used: a bank has a business process which takes the data from the client so that a transaction from the client's account to his credit card can be carried out which increases the credit of the card. The service should take a specific client's data and return the amount of an account, but due to fault this service returns the amount plus other data about another account. This service is therefore not working in compliance with the specification. If this fault is determined, it could be possible to throw a fault exception and, using compensation handlers, undo all transactions carried out from the moment of the fault. Compensation handlers can only be employed when an internal fault is detected. In the case described, the services were working well but the fault came from the functionality of the service.

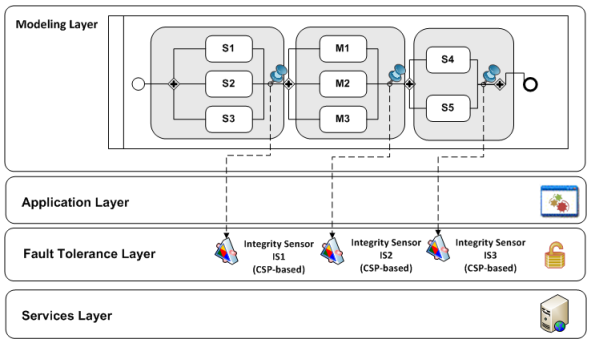



Figure 12. Approach with sensors already allocated.

Compensation is the really useful mechanism to undo transactions, but here it will be applied in another sense. In our case, when an sensor determines any fault in the execution of the business process, an exception is thrown at the end of that process. After throwing the fault, it is determined by the process and the compensation handler is invoked. Within of the compensation, the services with failures are then re-executed from a backup those correct functionality has already been tested. This process is shown in the Figure 13, where the faulty activities has been marked with a red cross within the activities. One consideration has to be taken into account, for example if the service M1 has a fault and the service S5 waits for any data generated from M1, there is a dependency between M1 and S5, therefore when M1 has a fault, then S5 has to be re-executed. In the scope of fault tolerance this solution is not purely a solution based on checkpointing since the state of the process is not saved and the execution continues from the checkpoint. The checkpointing approach does not use the same process because diagnosis is distributed from various sensors.

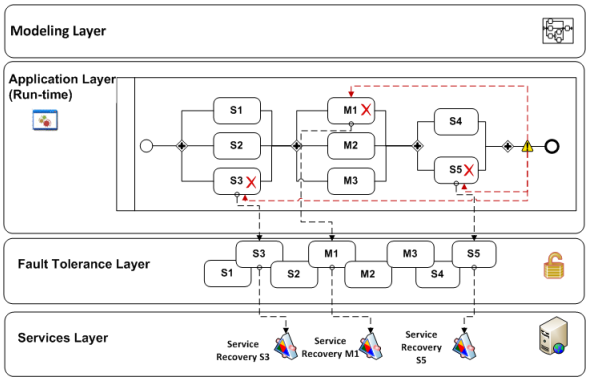



Figure 13. Recovery business process.

How does the framework work? Firstly, when the framework obtains an output, it is checked by consulting the oracle. If the oracle returns KO as response, then the framework attempt to isolate those components involved in the fault. To isolate faulty services, model-based diagnosis is employed using CSPs. In order to automate the process of diagnosis, a CSP model is solver together with the business process output. Once diagnosis retrieves which components are producing faults, then this information is translated to the fault tolerance layer and the corresponding mechanism is activated to replace erroneous services with other correct services. This process is outline depicted in Figure 14, and it is common to binder, binder backup and to NVP.

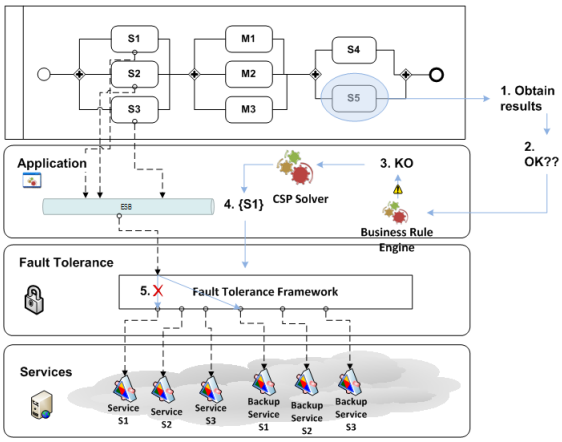
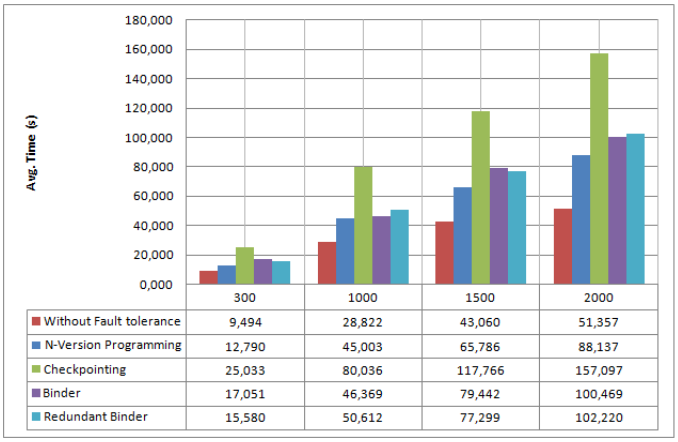
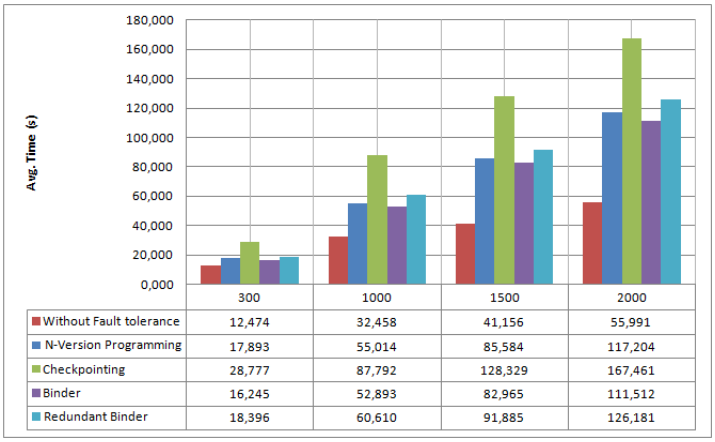



Figure 14. Example of the framework execution execution.




| | 300 | 1000 | 1500 | 2000 |
|---|---|---|---|---|
| Without Fault tolerance | 9,494 | 28,822 | 43,060 | 51,357 |
| N-Version Programming | 12,790 | 45,003 | 65,786 | 88,137 |
| Checkpointing | 25,033 | 80,036 | 117,766 | 157,097 |
| Binder | 17,051 | 46,369 | 79,442 | 100,469 |
| Redundant Binder | 15,580 | 50,612 | 77,299 | 102,220 |




| | 300 | 1000 | 1500 | 2000 |
|---|---|---|---|---|
| Without Fault tolerance | 12,474 | 32,458 | 41,156 | 55,991 |
| N-Version Programming | 17,893 | 55,014 | 85,584 | 117,204 |
| Checkpointing | 28,777 | 87,792 | 128,329 | 167,461 |
| Binder | 16,245 | 52,893 | 82,965 | 111,512 |
| Redundant Binder | 18,396 | 60,610 | 91,885 | 126,181 |

Figure 15. I. Performance one thread for execution. II. Performance more than one thread for execution.

## V. EVALUATION AND RESULTS

A set of test cases has been executed for each case of the fault-tolerant approach, as described in previous sections. The tests developed have been separated in two groups: first considering one thread of execution, and second one considering more than one thread of execution. Also on, for each thread a number of invocations 300,1000,1500 and 2000 are carried out. The tests simulate the idea of there be an integrity fault in a single component for each request, for instance *S1*. For each input of the process, a set of random value tests has been created. Although many parameters could be useful in order to measure dependability properties, in our proposal the most interesting in the comparison is the performance time. Performance give a clear indication about the difference between one solution and another to be measured in terms of deliver. Figure 15 shows two graphics which contain results about average performance time . The hardware used in the execution of the tests is a server Intel Xeon E5530 2.4 GHz, with 8GB RAM and a Debian Gnu/Linux 64bits OS and a client Intel Core 2 Duo T9300 2,5GHz with 4 GB RAM.

Taking the performance parameter into account, the binder solution obtains the best result (Figure 15), although, in this case, the performance could be affected by the diagnosis stage time. In order to avoid the utilization of a binder solution, we can opt for a checkpointing approach, but would then need to value not only the time required to locate sensors, but also the overhead for checking and extra work designing handlers. On the other hand, in order to avoid the diagnosis stage, we might opt for an NVP solution. In the development sense, time spent on developing correct components using NVP must be balanced against or a solution with binder using only a primary-backup solution. In the case of multiple faults, the binder solution may be insufficient for the replication (primary and backup) of services to ensure the correction of services. However, with NVP components and the correct number of replicas we may achieve a result with a very high level of correctness.

## VI. RELATED WORK

Dependability is studied in the context of business process management in [36], and a framework entitled *Dynamo* is presented. This framework provides a run-time business process supervisor that guarantees that the requirements of dependability are satisfied. The main contribution is the definition of two languages, WSCoL and WSRS, although they are not a supported standard. Likewise, [36] presents some remedial strategies that are mainly focused on the recovery context, but fail to pay attention to the typical solutions in the fault tolerance scope.

In the scope of fault tolerance for BPEL processes and Web Service composition, there are many contributions [37][38][39][40]. The feasibility of BPEL processes to implement fault tolerance techniques with BPEL language is studied in [37]. The work presents a tool for mapping fault tolerance techniques using BPEL language concepts and elements but it fails to show any example of an application or data tests with real conclusions for the work. Middleware to integrate some remedial strategies to handle violation constraint faults in the BPEL processes was developed in [39]. Whereby a framework structured in various components is developed. The most relevant components are: composition, analysis and instrumentation. The composition composes services and “business goals”, the analysis form the business process with a remedial strategy using remedial databases, and the last component, that of instrumentation, translates the process into a final BPEL process. Another studies is focused on the dynamic selection of Web Services for the construction of optimal workflows, [38]. The selection of the optimal service is based on searching from services from various repositories and data stored in databases. Although, this technique appears to provide a fault tolerance solution, this is solely due to the workflows being built on the fly

through the selection of the best service each time, but it fails to take into account the unique point of fault in the proxy component. However, we have provided a solution with a binder, and for the case of a faulty binder, we have developed another solution with a redundant binder. An architecture for self-healing BPEL processes is presented in [40]. Self-healing properties imply the development of many elements integrated into the same engine such as a monitor, a diagnoser, a planner of changes, and validation mechanisms. Some recovery strategies have been adopted in this work and integrated into the engine. Our work is more focused on mechanisms for fault-tolerance solutions, without using monitors or planners. In this sense, we have adopted an innovatory solution based on business rules (such as oracle) and CSP-based fault diagnosis.

There are some initiatives into introduction of fault-tolerance techniques in the area of Web Service, [41][42]. In these studies, the main contributions are the definition of a framework or middleware to achieve fault-tolerant service platforms. A fault-tolerance architecture for SOAP protocol is proposed and the solution is compared against the flexibility of CORBA solutions, [41]. In [42], the authors have defined and developed a mechanism to improve resilience to faults for Web service clusters to enhance the reliability of the services.

The majority of fault tolerance solutions are based on replication and recovery techniques. Although the replication is a very important concept in fault-tolerant systems, when it is necessary to create fault-tolerance in software, the solution of replication is insufficient. The philosophy in fault tolerance software is totally different; the techniques are based mainly on software diversity and data diversity, [18][34].

In fault tolerance of distributed systems, checkpointing and rollback recovery approaches are popular [43][31][44]. A well-designed checkpointing algorithm allows a faulty business process to recover the recently saved state. Further proposals have been developed in other domains such as grid computing [45] and Web Services [36].

## VII. CONCLUSION AND FUTURE WORK

In this paper, an innovative framework for the development of business processes with dependable capabilities based on fault tolerance has been introduced. The framework is composed of three main elements: business rules, model-based diagnosis based on constraint programming, and fault-tolerant mechanisms. The innovation in this framework is the use of a business rule engine as an oracle of solutions, and model-based diagnosis to automate the determination and isolation of components using CSP techniques in the case of a fault. In the sense of fault tolerance, the framework presents various solutions: the first solution create fault-tolerant operational business processes (BPEL) using dynamic binding techniques and replication of services; the second solution presents an improvement introducing replication of the binder; a third solution uses the concept of software tolerance and implements NVP components; and a fourth solution provide a simulation of a checkpoint approach. To the best of our knowledge, this work is the first contribution with fault tolerance based on software fault-tolerance and checkpointing approaches for business processes.

As future work, it could be interesting to add new features of fault tolerance within the framework. The work will be extended to include with other software fault tolerance techniques such as as Recovery Block, or to introduce new features into the N-Version components, for example, changing the complexity of the adjudicator or studying the number of variations in function in terms of the service. Likewise, the framework could be extended to embrace other capabilities to achieve self-healing and self-adaptability approaches.

## ACKNOWLEDGEMENTS

This work has been partially funded by the Department of Innovation, Science and Enterprise of the Regional Government of Andalusia project under grant P08-TIC-04095, by the Spanish Ministry of Science and Education project under grant TIN2009-13714, and by FEDER (under the ERDF Program).

## REFERENCES

[1] A. J. Varela-Vaca, R. M. Gasca, D. Borrego, and S. Pozo, "Towards dependable business processes with fault-tolerance approach," in *Proceedings of the 2010 Third International Conference on Dependability*, ser. DEPEND 2010. IEEE Computer Society, 2010, pp. 104–111.

[2] A. J. Varela-Vaca and R. M. Gasca, "Opbus: Fault tolerance against integrity attacks in business processes," in *Computational Intelligence in Security for Information Systems 2010*, ser. Advances in Intelligent and Soft Computing, vol. 85. Springer Berlin / Heidelberg, 2010, pp. 213–222.

[3] Gartner Inc. Report, "Gartner EXP worldwide survey of nearly 1.600 CIOs shows IT budgets in 2010 to be at 2005 levels," 2010. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1283413

[4] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.

[5] W. M. P. van der Aalst, A. ter Hofstede, and M. Weske, "Business process management: A survey," in *Proceedings of the 1st International Conference on Business Process Management*, ser. Lecture Notes in Computer Science, vol. 2678. Springer Berlin / Heidelberg, 2003, pp. 1019–1031.

[6] A. J. Varela-Vaca, R. M. Gasca, and L. Parody, "OPBUS: Automating Structural Fault Diagnosis for Graphical Models in the Design of Business Processes," in *21th International Workshop in Principles of Diagnosis (DX'10)*, 2010, pp. 337–341.

[7] S.-M. Huang, Y.-T. Chu, S.-H. Li, and D. C. Yen, "Enhancing conflict detecting mechanism for web services composition: A business process flow model transformation approach," *Information Software Technology*, vol. 50, no. 11, pp. 1069–1087, 2008.

[8] J. Mendling, M. Moser, G. Neumann, H. M. W. Verbeek, and B. F. Vandongen, "Faulty EPCs in the SAP Reference Model," in *International Conference on Business Process Management (BPM 2006)*. Springer-Verlag, 2006, pp. 451–457.

[9] Condor Team, "Condor project," 2010. [Online]. Available: http://www.cs.wisc.edu/condor/

[10] C. Pautasso and G. Alonso, "Flexible binding for reusable composition of web services," *Software Composition*, vol. 3628/2005, no. 1820, pp. 151–166, 2005.

[11] P. Neira, R. M. Gasca, and L. Lefèvre, "Demystifying cluster-based fault-tolerant firewalls," *IEEE Internet Computing*, vol. 13, no. 6, pp. 31–38, 2009.

[12] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella, "Automatic service composition based on behavioral descriptions," *International Journal of Coorporative Information Systems*, vol. 14, no. 4, pp. 333–376, 2005.

[13] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. Buchmann, "Extending BPEL for run time adaptability," in *Ninth IEEE International EDOC Enterprise Computing Conference EDOC05*. IEEE Computer Society, 2005, pp. 15–26.

[14] J. De Kleer and J. Kurien, "Fundamentals of model-based diagnosis," *Fault detection supervision and safety of technical processes*, pp. 25–36, 2004.

[15] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transaction on Dependable and Secure Computating*, vol. 1, no. 1, pp. 11–33, 2004.

[16] L. Liu, Z. Wu, Z. Ma, and Y. Cai, "A dynamic fault tolerant algorithm based on active replication," in *7th International Conference on Grid and Cooperative Computing (GCC'08)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 557–562.

[17] R. Baldoni, C. Marchetti, and S. T. Piergiovanni, "Asynchronous active replication in three-tier distributed systems," in *Proceedings 9th IEEE Pacific Rim Symposium on Dependable Computing (PRDCF'02)*, 2002.

[18] L. L. Pullum, *Software fault tolerance techniques and implementation*. Norwood, MA, USA: Artech House, Inc., 2001.

[19] Object Management Group (OMG), "Business process model and notation," 2009. [Online]. Available: http://www.omg.org/spec/BPMN/1.2

[20] T. Debevoise, *Business Process Management with a Business Rules Approach: Implementing the Service Oriented Architecture*. Business Knowledge Architects, 2005.

[21] W. M. van der Aalst and A. H. M. Ter Hofstede, "Verification of workflow task structures: A petri-net-based approach," *Information Systems*, vol. 25, pp. 43–69, 2000.

[22] W. Sadiq, Maria, and E. Orlowska, "Analyzing process models using graph reduction techniques," *Information Systems*, vol. 25, pp. 117–134, 2000.

[23] H. Lin, Z. Zhao, H. Li, and Z. Chen, "A novel graph reduction algorithm to identify structural conflicts," in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, vol. 9. Washington, DC, USA: IEEE Computer Society, 2002, pp. 289–299.

[24] OASIS, "Business Process Execution Language," 2008. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[25] C. Ouyang, W. M. P. Van Der Aalst, and M. Dumas, "Translating BPMN to BPEL," *Business*, vol. 2006, pp. 1–22.

[26] S. A. White, "Using bpmn to model a bpel process," Tech. Rep., 2006.

[27] D. Chappell, *Enterprise Service Bus*. O'Reilly Media, Inc., 2004.

[28] CHOCO Team, "Choco: an open source java constraint programming library," 2010. [Online]. Available: http://www.emn.fr/z-info/choco-solver/pdf/choco-presentation.pdf

[29] D. Borrego, R. Gasca, M. Gomez, and I. Barba, "Choreography analysis for diagnosing faulty activities in business-to-business collaboration," in *20th International Workshop on Principles of Diagnosis. DX-09*, Stockholm, Suecia, 2009, pp. 171–178.

[30] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*. Elsevier, 2006.

[31] J. L. Kim and T. Park, "An efficient protocol for checkpointing recovery in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 8, pp. 955–960, 1993.

[32] A. Erradi and P. Maheshwari, "Dynamic binding framework for adaptive web services," in *ICIW '08: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 162–167.

[33] U. Küster and B. König-Ries, "Dynamic binding for BPEL processes - a lightweight approach to integrate semantics into web services," in *Second International Workshop on Engineering Service-Oriented Applications: Design and Composition (WESOA06) at 4th International Conference on Service Oriented Computing (ICSOC06)*, Chicago, Illinois, USA, 2006, pp. 116–127.

[34] W. Torres-Pomales, "Software fault tolerance: A tutorial," Tech. Rep., 2000.

[35] D. Borrego, M. T. Gomez-Lopez, R. M. Gasca, and R. Ceballos, "Determination of an optimal test points allocation for business process analysis," in *2010 IEEE/IFIP Network Operations and Management Symposium Workshops (BDIM 2010)*, 2010, pp. 159–160.

[36] L. Baresi, S. Guinea, and M. Plebani, "Business process monitoring for dependability," in *Proceedings of the Workshops on Software Architectures for Dependable Systems (WADS'06)*, 2006, pp. 337–361.

[37] G. Dobson, "Using ws-bpel to implement software fault tolerance for web services," in *EUROMICRO-SEAA*, 2006, pp. 126–133.

[38] L. Huang, D. W. Walker, O. F. Rana, and Y. Huang, "Dynamic workflow management using performance data," in *IEEE International Symposium on Cluster, Cloud, and Grid Computing*, 2006, pp. 154–157.

[39] M. Wang, K. Y. Bandara, and C. Pahl, "Integrated constraint violation handling for dynamic service composition," in *SCC '09: Proceedings of the 2009 IEEE International Conference on Services Computing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 168–175.

[40] S. Modafferi, E. Mussi, and B. Pernici, "Sh-bpel: a self-healing plug-in for ws-bpel engines," in *MW4SOC '06: Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*. New York, NY, USA: ACM, 2006, pp. 48–53.

[41] C.-L. Fang, D. Liang, F. Lin, and C.-C. Lin, "Fault tolerant web services," *J. Syst. Archit.*, vol. 53, no. 1, pp. 21–38, 2007.

[42] M.-Y. Luo and C.-S. Yang, "Enabling fault resilience for web services," *Computer Communications*, vol. 25, no. 3, pp. 198 – 209, 2002.

[43] G. Cao and M. Singhal, "Checkpointing with mutable checkpoints," *Theoretical Computer Science*, vol. 290, no. 2, pp. 1127–1148, 2003.

[44] R. Baldoni, "A communication-induced checkpointing protocol that ensures rollback-dependency trackability," in *FTCS '97: Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 68–77.

[45] X. Shi, J.-L. Pazat, E. Rodriguez, H. Jin, and H. Jiang, "Adapting grid applications to safety using fault-tolerant methods: Design, implementation and evaluations," *Future Generation Computer Systems*, vol. 26, no. 2, pp. 236 – 244, 2010.