2010 Third International Conference on Dependability

Towards Dependable Business Processes with Fault-Tolerance Approach

Angel Jesus Varela-Vaca, Rafael M. Gasca, Diana Borrego, Sergio Pozo
Departamento de Lenguajes y Sistemas Informáticos,
ETS. Ingeneiría Informática, Avd. Reina Mercedes S/N,
Universidad de Sevilla,
Sevilla, Spain
{*ajvarela, gasca, dianabn, sergiopozo*}*@us.es*

***Abstract*—The management and automation of business processes have become an essential tasks within IT organizations. Companies could deploy business process management systems to automatize their business processes. BPMS needs to ensure that those are as dependable as possible. Fault tolerance techniques provide 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. The framework presents different solutions in the fault tolerance scope. The solutions have been developed using a practical example and some results have been obtained, compared and discussed.**

***Keywords*-business process; fault-tolerance; dependability; reliability**

I. INTRODUCTION

In the last 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 modeling, design, administration, configuration, enactment and analysis of business processes [1]. A business process model is a set of activities that are executed in coordination within an organizational and technical environment to realize a set of business goals.

BPM has turned into an essential tool for organizations. BPM as methodology pursues to improve the efficiency through systematic management of business processes that should be modeled, automatized, integrated, monitored and optimized in a continue form. One of the most important goals of BPM is the better understanding of the operations a company performs and the relationships among these operations. BPM also aims at narrowing the gap between business processes that a company performs and the implementation of these processes in Business Process Management System (BPMS). BPMS is a set of software tools to manage business processes.

Companies could deploy BPMS to automatize their business process, but they have to ensure that those are as dependable as possible. Take into account the dependable process operation is a significant requirement for many types of companies: electronic banking and commerce, automated manufacturing, etc. The cost and consequences of failures of these systems range from mildly annoying to catastrophic, with serious injury occurring or lives lost, systems destroyed, security breaches, and so on.

BPM paradigm follows a life cycle which consists in several stages [2], shown in Figure 1. During the stages, different kinds of faults can be introduced:

- In the design stage, business process models can present some design faults (such as deadlocks, live locks or starvations). Some systematic approaches have been provided design guidelines for designers that allow them to correct and improve their designed processes. Design problems are not taken into account in this paper because it is a problem that has already been discussed [3] [4].
- In the enactment stage, output process faults (verification) can be generated for business processes when a business process obtains an unexpected output, unexpected message, unexpected events or also an unexpected performance. Executable business processes usually use external services that are not under our jurisdiction. Thus, it is not possible to ensure that change of functionality appears during the business process life cycle.

Therefore, the inclusion of measures that allow to reduce the risk of fault and increase the dependability of business processes from design stages it will be necessary. The approach proposes to achieve more dependable business processes based on fault tolerance. Some proposals have used the fault tolerance ideas in grid computing, composition of applications or service-oriented architectures, [5] [6] [7] [8]. In these works, different fault tolerance approaches have been applied: check-point view [5], recovery techniques [6], and other sophisticated techniques such as dynamic binding [7], and self-reconfiguration of systems [8]. Our proposal is based on the classical fault tolerence ideas of replication, but introducing other necessary elements like dynamic binding, and providing techniques of diversity in the software fault tolerance scope.

This paper is structured as follows: Section II introduces some concepts of BPM and fault tolerance; Section III presents a framework for dependable business processes using fault-tolerant techniques; Section IV a practical exam-

978-0-7695-4090-0/10 $26.00 © 2010 IEEE
DOI 10.1109/DEPEND.2010.24

ple is developed; Section V shows experimental results that are discussed; and, in the last section, different conclusions and future work are provided.

II. Business Process Management System and Fault Tolerance

In order to understand BPM(S), it is necessary to show the typical business process life cycle [2], shown in Figure 1. Life cycle consists in different stages:

1) **Process design:** business process models are defined.
2) **System configuration:** a BPMS is chosen and configured.
3) **Process enactment:** the processes are deployed in a BPMS.
4) **Diagnosis:** techniques are applied to identify and isolate faults in business processes. Most diagnosis techniques applied are focused on identifying design faults.

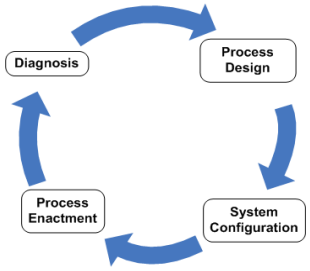

Figure 1. Business process life cycle.

In this paper, only two levels of BPM are considered [1]:

- 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.
- 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 use external services that are not under our jurisdiction. This means that it is not possible to ensure that, during BPM life cycle, the services will not change in functionality or that they will have faults or not suffer attacks, etc.

A failure is an event that occurs when the delivered service deviates from correct behaviour. An error is the part of the system state that may cause a subsequent failure. A fault is identified as the hypothesized cause of an error, sometimes called ”bugs”. Faults can occur due to malfunction or design errors of software or hardware [9].

Fault diagnosis permits to determine why a business process correctly designed does not work as it is expected [10]. The diagnosis aims to identify and isolate the reason of an unexpected behavior, or in other words, to identify the parts which fail in a business process. Although diagnosis is used in this work to isolate the faulty services, this topic is not the central scope of this paper.

The proposal is focused on achieving dependable business processes but means to achieve dependability [9]. The meaning to achieve dependability properties falls into major groups: fault tolerance, fault avoidance or prevention, fault removal, fault forecasting.

By definition, fault tolerance [9] is a mechanism used to guarantee service complying with the specification in spite of faults. Fault-tolerance techniques are a manner to reduce the risks of faults. Fault-tolerance frameworks are based on replication of components. The replication concept aims the recovery of services by replicating each of its functionality in some replicas. Typical solutions in replication are considered as:

- *Passive replication* [11]: the client only interact with one replica (primary), it handles the client request and sends backs response. The primary also issues messages to the backup replicas (other replicas) in order to update their state.
- *Active replication* [11]: all replicas play the same role. All of them receive each request, handle it the request, and send back the response to the client. There exist other solutions based on active replication [12].

At first time, active replication provides generally faster response time than passive replication. However, in active replication all replicas should to process the requests, so it needs more system resources than passive replication. Moreover, the nested redundant replicas could arise an invocation problem while using active replication, requiring the replicas to be deterministic. The passive replication does not require that determinism, and for this reason it is more flexible.

Increasing the dependability of software presents some unique challenges when it is compared to traditional hardware systems [13]. Hardware faults are mainly physical ones, 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, simply to add redundancy is not enough, as it is typically done for hardware faults, because doing so will simply duplicate the problem.

III. Fault Tolerance Framework

In the previous section, the basic ideas of business process life cycle were introduced and the main stages to manage business processes were shown. The proposed framework is based on the ideas of BPM life cycle, and its structured is depicted in Figure 2.

The main characteristic is the use of fault tolerance techniques to mitigate the risk of possible process faults. In the next subsections the different parts and details of the framework are introduced and discussed.

In the proposal it is necessary to do some assumptions before continuing:

1) A business process model has a single start event, a single end condition and every tasks contribute to finish the process in a correct form.
2) There not exist design faults (deadlocks, live locks, starvations, etcetera).
3) Operational business processes can not suffer Byzantine faults (arbitrary faults).
4) Operational business processes are stateless.

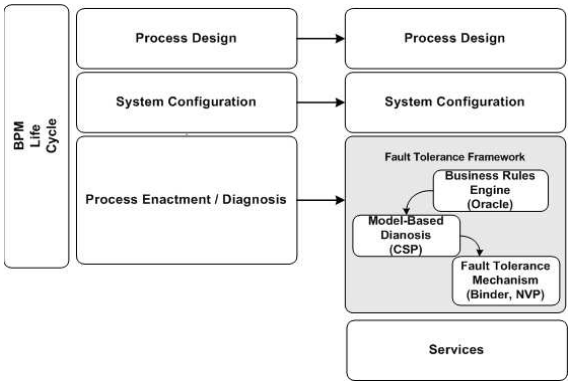

Figure 2. General view of the framework.

Therefore, the main problems to discuss in this paper are to diagnose and mitigate incorrect outputs/results and events in business processes. In the next sub-sections every framework layer is going to be described.

A. Process Design

Our approach is focused around three kinds of models:

- Business rule model: a business rule is a statement that defines some aspects of the business processes that are not possible to express in the model [14]. These rules can be formalized as "if-then" statements for selected Business Rule Management systems (BRMS) or use natural language or formal methods. BRMS is a software system used to define, deploy, execute, monitor and maintain the variety and complexity of decision logic that is used by operational systems within an organization. Business rules will act as an "oracle". That is, business rules indicate the correctness of the outputs from the business processes.
- Business process model describes a business process (BP). BPs are usually defined with graphical languages (simply diagrams) but nowadays it is necessary to narrow the gap between business stakeholders (designers, annalists, developers, etc). Operational business processes usually are represented in Business Process Management Notation (BPMN) [15]. BPMN contains different elements such as activities, data objects, gateways, etc. Business Process Execution Language (BPEL) is a de facto standard language to implement service-based business processes and lots of commercial tools support it. Although BPMN is a standard to design process models, they can be automatically translated into BPEL [16] [17], and some commercial BPMS makes translations from BPMN diagrams to BPEL processes. In the proposal, BPEL has been selected to represent the operational processes.
- Constraint model describes the behavior of the business process model using constraints. The constraint model is necessary for the diagnosis stage where Constraint Satisfaction Problems (CSP) are used. CSPs will be described in Section III-C. Constraint model can be constructed off-line or on-the-fly. Process constraint models can be built online because logs can be captured where information of inputs, outputs and service constraint model can be given.

B. System Configuration

In this layer, different technologies which are used to implement and deploy models are introduced. The main advantages to design business processes with BPEL are: BPEL has necessary elements to implement fault tolerant mechanisms; it is a service-based process language; some of the distinguished commercial tools of BPMS (Intalio, Enterprise Architect, Borland Together, Oracle BPMS, etc.) and other non-commercial (Netbeans, Eclipse BPEL, jBPM, etc.) support BPEL. Therefore, BPEL processes are going to be used for the proposal. A BPEL environment is necessary to implement and deploy operational business process. In this proposal, NetBeans together with GlassFishESB have been selected. NetBeans provides an integrated design environment to develop graphically business processes defined in BPEL. GlassFishESB is an application server with several components to support functions of Enterprise Service Bus (ESB) [18]; it also has a runtime service to execute BPEL processes.

For business rules, there is not an accepted standard for business rules systems. For this reason, in this proposal, WebSphere Ilog JRules has been selected.

For diagnosis, any CSP Solver could be used. In this case, ChocoSolver has been selected. The model has been implemented using the Choco constraint programming API, although the model could be implemented standalone from other CSP Solver tools (for instance JACoP) and used as service.

C. Diagnosis

In the diagnosis stage, model-based diagnosis is applied. The diagnosis will only be invoked when a fault in the business processes outputs has been detected. In order to diagnose, Constraint Satisfaction Problems (CSP) techniques have been applied. CSP is an extended form to solve problems [19] [20]. The constraint model designed, in the design stage, is a transformation of the business process to

Constraint Optimization Problem (COP). This model can be deployed in a CSP Solver tool, for instance ChocoSolver, and the model together with the obtained outputs are used to identify and isolate possible behavioral faults in the components.

D. Fault Tolerance Mechanisms

On the one hand, our proposal adopts some replication solutions to obtain fault tolerance in BPEL processes. On the other hand, software fault tolerance techniques are provided, introducing the concept of diversity. Diversity means to provide identical services but with separating design from implementation. The techniques used in the framework are described as follows:

1) **Without fault tolerance**
   BPEL processes are defined as composition of services. Therefore, the web services (external or internal) are linked at design time. That is, if a fault output or event occurs (supposing not design errors within processes) this fault is located in the service side, so it is not possible to replace the faulty service at run-time. It is only possible at design time.
   The faults only will be solved stopping every business processes instances. After that, the faulty service has to be located (diagnosis), eliminated and replaced for another correct service. Therefore, it is necessary to introduce mechanisms to mitigate this effect 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 case of fault, it is possible to use the backup services. The adopted solution in the proposal is based on dynamic binding ideas [21] [22]. Dynamic binding is a technique that allows to link services at run-time.
   At first approximation, a binder component has been introduced between the BPEL processes and the services acting as proxies, Figure 3. The binder component is not developed as external program or external monitor, but as another process. The binder component may decide at run-time what services to invoke: if the original service or the replicas (backups). The solution is fault tolerant because, in case of detecting a faulty service, every faulty service invocations have to be replaced for a backup service invocation.
   This is a good solution, but presents some deficiencies such as an unique point of fault has been included (binder), and the complexity of the binder can introduce a very high overhead in the performance. In order to solve the first one, the replication of binder can be applied. That is, replicating the binder component (one primary and backups replicas). In case of fault,

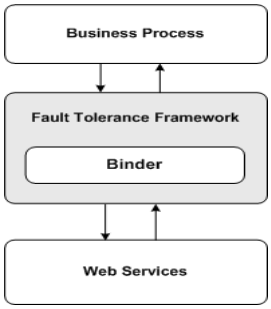

Figure 3. Communication process-service with binder.

in the binder component, the backup replica takes the control of the execution, and continues executing, Figure 4. The binder backup can be an exact copy of the original. BPEL provides different mechanism, like faults and compensate handlers, to achieve the implementation of primary-backup binders. In our case, passive replication is used in the proposed solution.

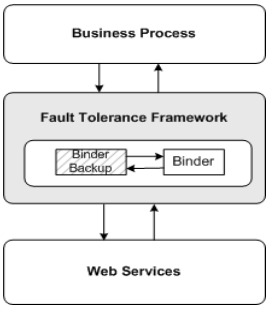

Figure 4. Communication process-service with binder replicated.

3) **Multi-Version (N-Version Programming) approach**
   The diversity is a very important factor to aim dependable software systems. The main goal of diversity is to provide identical services (variants) but with separated design and implementation to minimize the causes of identical faults. For instance, when a software variant presents a fault, this will be as isolated as possible. There are different techniques for fault-tolerance software based on multi-version (diversity of software) [13] [23]: N-Version Programming (NVP), Recovery Blocks, N-Self Checking Programming.
   In the proposed framework, one adopted solution is focused on NVP, which is a static technique where a task is executed for various processes or programs and the result is only accepted for majority of votes. NVP uses different adjudicator/decision mechanism (DM) [13].
   The paradigm of N-Version considers to use diversity for implementations and designs to isolate faults in the components. The different services that are used will be a N-Version Component. The selected implementation for N-Version components are following the basic ideas of N-Version Programming. N-Version component provides at most *2X + 1* replicas, where

X can vary from *1* to N, and where N can vary 1, 2, 3, or more. Components have been developed with an adjudicator (DM) to obtain the output results, Figure 5. The logic within adjudicator (DM) can be very basic or really complicated. However, NVP components can be improved by adding new features to the adjudicator or developing new strategies, but introducing more design complexity in the N-Version components probably introduces high overhead in the performance of the processes.

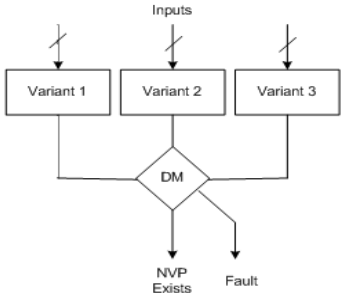

Figure 5. Example of N-Version component.

Using N-Version components fault-tolerance is achieved, but it supposes another advantage since the diagnosis stage is not necessary at all. For example, if one of the variants does not return a response in time, the adjudicator takes the results from the other variants. In consequence, the diagnosis can be eliminated from the framework, although it could introduces a very high cost in developing and performance.

IV. EXAMPLE OF APPLICATION

In order to clarify the problematic, an example has been developed. Although it is a simple example with simple operations, it can illustrate perfectly the problematic discussed. 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 other five elements with the same functionality *S(x)* but also independent, Figure 6.

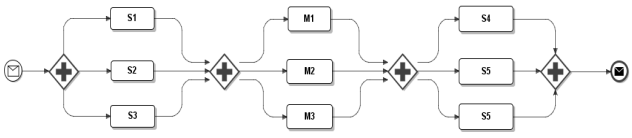

Figure 6. Example of business process.

To illustrate a real example, the process has been distributed. The global process has been divided into three different processes and they have been deployed in three different systems. The new distributed process is shown in Figure 7. In this case, there is a global process; *"Complete 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.

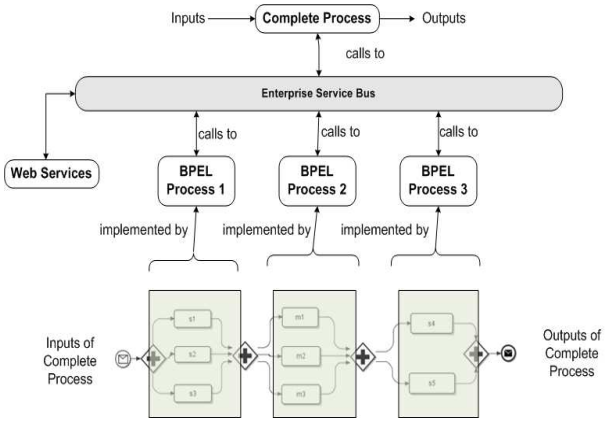

Figure 7. Distributed process.

In this scenario, it is possible to observe different aspects. For instance, a fault within service *S1* has directly an effect to the BPEL Process 1 result, as consequence the BPEL Process 2 is affected and finally BPEL Process 3 will be affected, so the final result of the complete process will be not correct. In order to improve the fault tolerance in the process, it is necessary to insert fault tolerance mechanisms such as previously introduced.

1) **Solution Primary-Backup Services with unique binder.** This solution applies the concept of redundant services with dynamic binding mechanisms. This approximation has a primary service as principal and one or more backups. In case of fault diagnosis of a service, it will be possible to replace the invocations from this primary to the backup service. A binder component has been introduced between the process and the services, Figure 8. The binder component is common for every BPEL process. The solution is fault-tolerant but introduces a unique point of fault as binder component.

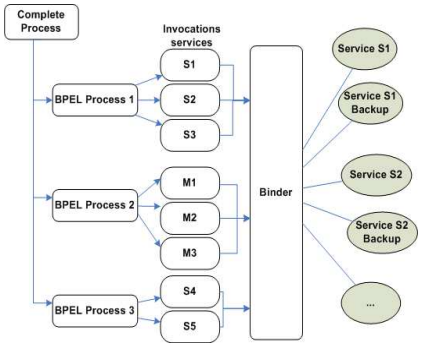

Figure 8. Fault-Tolerant solution with binder.

2) **Solution Primary-Backup Services with replicated binder.** In this case a replication of the binder is introduced. If the binder component enters in fault state, the backup replica can take the control of the execution, Figure 9. In the developed proposal, the binder backup is an exact copy of the original. The binder component is common for every BPEL process.

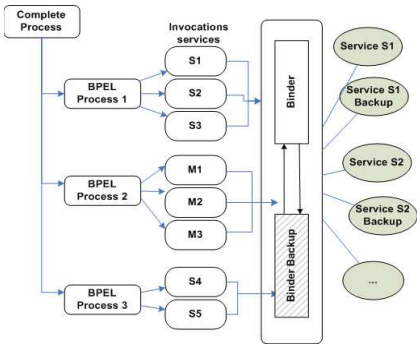

Figure 9. Fault-Tolerant solution with replicated binder.

3) **Solution N-Version components.** The N-Version components developed use three variants with an adjudicator to obtain the output results (Figure 10). The logic within adjudicator is basic. The adjudicator is basic to do not introduce high overhead in the final performance of the process. Therefore, it only compares the outputs from the different variants by means of a voting system. In this implementation, fault-tolerant software component is achieved and the diagnosis stage is not necessary at all.

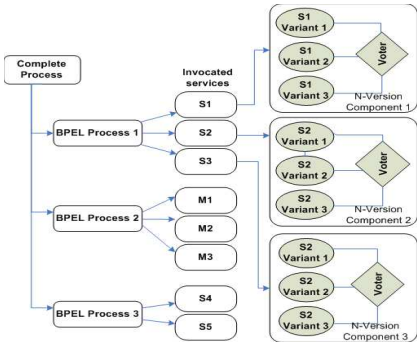

Figure 10. Fault-Tolerant solution with N-Version components.

Table I provides a comparative of the fault tolerance applied:

- Type. What kind of fault tolerance has been used. For example, the replication of services in the case of binder.
- Diagnostic. If the diagnosis is necessary or not for this technique. For example, in NVP solution the diagnosis stage is not necessary.
- Overhead. It indicates the additional logic introduced to develop this technique. For example, in the case of binder a dynamic binding additional logic is necessary.

V. EVALUATION AND RESULTS

A set of test cases has been executed for each case of fault-tolerant approach which has been described in previous sections. Developed tests use three hundred requests for each solution. The tests are based on the idea of having a integrity fault in single component for each request, for instance *S1*. For each input of the proccess, a set of random value tests have been created where the domain of the inputs runs from 0 to N (N is the maximum value of an integer in Java language). The hardware used to execute the tests is a server Intel Xeon E5530 2.4 GHz, with 8GB RAM and Debian Gnu/Linux 64bits OS.

Different parameters are important to try to achieve dependability in business processes. Dependability involves many parameters. For our proposal, it is more interesting to measure how good are the applied fault tolerance mechanisms. All measures together will give an idea of how much dependability has been achieved:

- *The performance* is an interesting parameter that allows to measure the difference between one solution and another in terms of delivery.
- *Diagnostic time* to identify the faulty service can provide an idea of overhead identifying the fault.
- *Average of number of used services* is another interesting parameter to measure. For example, in the case of N-Version, it presents a very high overload in terms of the services used.
- *Number of requests* that are executed.

Taking into account the performance parameter, the binder solution obtains the best result (Figure 11). Although, in case of binder solution the performance could be affected for the diagnosis stage time. If we want to avoid the diagnosis stage, we might opt for a NVP solution. In the development sense, we must balance if we want to spend more on developing correct components using NVP or a solution with binder using only primary-backup solution. In case of multiple faults, the binder solution might be not enough, so that the replication (primary and backup) of services do not ensure the correction of services. However, with NVP components and a correct number of replicas we might achieve a result with a very high level of correction.

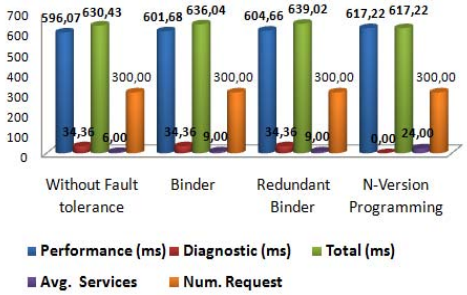

Figure 11. Experimental results.

Table I
COMPARATIVE OF THE FAULT TOLERANCE TECHNIQUES.

	Type	Diagnosis	Overhead	Comment
Without Fault Tolerance	*NA*	+	Cost repair	-
Binder	Replication (Primary/Backup)	+	Dynamic binding	Replication of the services. Introduce binder
Redundant binder	Replication (Primary/Backup)	+	Dynamic binding	Replication of the services and binder
N-Version Programming	Diversity (N-Version Component)	-	Adjudicator (Voting system)	-

VI. RELATED WORK

The dependability has been studied in the context of business process management in [24]. In this work, a framework is proposed, *Dynamo*. This framework provides a run-time business process supervisor that guarantees that the requirements of dependability are satisfied. The framework is based on BPEL processes, but the main contribution in this work the definition of two languages. The first one, WSCoL, is used to monitor processes, and the second one, WSRS, is used to define reaction strategies. These languages are not a supported standard. Also, a set of remedial strategies are provided mainly focused on the recovery context, but these do not pay attention in the typical solutions in the fault tolerance scope such solutions based on replication or software fault tolerance techniques like the ones applied in our work.

In the scope of fault tolerance in BPEL processes and Web Service composition, there are some contributions, [25] [26] [27] [28]. The feasibility of BPEL processes to implement fault tolerance techniques with BPEL language is studied in [25]. The work presents a tool for mapping fault tolerance techniques using BPEL language concepts and elements but it does not show any example of application or data tests with real conclusions for their work. A middleware to integrate some remedial strategies to handle violation constraint faults in the BPEL processes was developed in [27]. They have developed a framework structured in different components. The most relevant components are: composition, instrumentation and analysis. The composition composes services and "business goals", the analysis composes the business process with a remedial strategy using remedial databases, and the last one, the instrumentation component translates the process into a final BPEL process. Another work is focused on the dynamic selection of Web Services for the construction of optimal workflows, [26]. The selection of the optimal service is based on searching services from different repositories and some stored data from databases. Although, it seems a fault tolerance solution, it is only because they build the workflows on the fly selecting the best service each time, but it does not take into account the unique point of fault in the proxy component. However, we have provided a solution with a binder, but in case of faulty binder we have developed another solution with redundant binder. An architecture for self-healing BPEL processes is presented in [28]. Self-healing properties implies to develop many elements integrated in the same engine like a monitor, diagnosers, a planner of change and validation mechanisms. Some recovery strategies have been adopted in this work and integrated in the engine. Our work is more focused on introducing mechanisms for fault-tolerance solutions in BPEL processes, without using monitors or planners. In this sense, we have adopted an innovatory solution based on business rules (as oracle) and CSP-based fault diagnosis. In the other hand, the solution with NVP components does not need the diagnosis stage.

In the Web services side, there are some initiatives to introduce fault-tolerance techniques, [29] [30]. In these works, the main contributions are the definition of a framework or middleware to achieve fault-tolerant services platforms. A fault-tolerance architecture for SOAP protocol is proposed and the solution is compared with the flexibility in front of CORBA solutions, [29]. In [30], the authors have defined and developed a mechanism to realize a fault resilience 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 make fault-tolerance in software the solution of replication is not enough. The philosophy in fault tolerance software is different, the techniques are based mainly on the software diversity and data diversity, [13] [23].

VII. CONCLUSION AND FUTURE WORK

In this paper, a framework to develop dependable business processes based on fault tolerance has been introduced. The framework is composed of three main elements: business rules, CSP-based fault diagnosis and fault-tolerant mechanism. The innovation in this framework is using a business rule engine as oracle of solutions and CSP techniques to isolate faults in case of fault detection. In the sense of fault-tolerance, the framework presents different solutions: the first solution makes fault-tolerant operational business

processes (BPEL) using dynamic binding techniques and replication of services; the second solution presents an improvement introducing replication of binder; and the third solution uses the concept of software tolerance and implements NVP components. In order to clarify the problematic, an example has been developed and some results have been discussed.

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

ACKNOWLEDGEMENTS

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

REFERENCES

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

[2] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, *Business Process Management: A Survey*. Springer-Verlag, 2003.

[3] 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.

[4] 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.

[5] C. Team. (2009) Condor. [Online]. Available: http://www.cs.wisc.edu/condor/

[6] C. Pautasso and G. Alonso, "Flexible binding for reusable composition of web services," in *Software Composition*, 2005, pp. 151–166.

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

[8] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. P. Buchmann, "Extending bpel for run time adaptability," in *9th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2005)*, 2005, pp. 15–26.

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

[10] D. Borrego, R. Gasca, M. Gomez, and B. I., "Choreography analysis for diagnosing faulty activities in business-to-business collaboration," in *20th International Workshop on Principles of Diagnosis (DX-09)*, Stockholm, Suecia, 2009.

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

[12] 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 (PRDC02)*, 2002.

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

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

[15] OMG. (2009) Business Process Model And Notation. [Online]. Available: http://www.omg.org/spec/BPMN/1.2

[16] C. Ouyang, M. Dumas, S. Breutel, and A. H. M. T. Hofstede, "Translating standard process models to bpel," in *Advanced Information Systems Engineering - CAiSE 2006, Luxembourg, Grand-Duchy of Luxembourg*. Springer, 2006, pp. 417–432.

[17] S. A. White, "Using bpmn to model a bpel process," *BP-Trends*, vol. 3, no. 3, pp. 1–18, 2005.

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

[19] R. Ceballos, R. M. Gasca, and D. Borrego, "Diagnosing errors in dbc programs using constraint programming," in *Current Topics in Artificial Intelligence*. Springer Berlin / Heidelberg, 2006, pp. 200–210.

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

[21] 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.

[22] 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)*, Chicago, Illinois, USA, December 2006.

[23] W. Torres-Pomales, "Software fault tolerance: A tutorial," NASA Langley Research Center, Tech. Rep. NASA/TM-200-210616, 2000.

[24] L. Baresi, S. Guinea, and M. Plebani, "Business process monitoring for dependability," in *Workshop on Algorithms and Data Structures*, 2006, pp. 337–361.

[25] G. Dobson, "Using ws-bpel to implement software fault tolerance for web services," in *EUROMICRO-Conference of Software Engineering Advanced Application*, 2006, pp. 126–133.

[26] 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.

[27] 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.

[28] 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.

[29] C. L. Fang, D. Liang, F. Lin, and C. C. Lin, "Fault tolerant web services," *Journal of Systems Architecture*, vol. 53, no. 1, pp. 21–38, 2007.

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