OPBUS: Fault Tolerance Against Integrity Attacks in Business Processes

Angel Jesus Varela Vaca and Rafael Martínez Gasca

Abstract. Management and automation of business processes have become essential tasks within IT organizations. Nowadays, executable business processes are the most extended kind of operational business processes. They usually use external services which are not under our jurisdiction and an intrusion attack over them could be not controlled, introducing unexpected fault in their execution. Organizations must ensure that their business processes are as dependable as possible before automating them. Fault tolerance techniques provide certain mechanisms to decrease the risk of possible faults in systems. In this work, OPBUS framework is proposed as solution for developing business processes with fault tolerance capabilities. Fault tolerance techniques are applied to resist faults related with integrity attacks over services involved in business processes. To the best of our knowledge, this work is the first approach that achieves a fault tolerance solution over business processes based on checkpoints and rollback using constraint programming.

1 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 [19]. BPM has turned into an essential tool in the current organizations. BPM aims at narrowing the gap between business processes that a company performs and the implementation of these processes in

Angel Jesus Varela Vaca · Rafael Martnez Gasca
Departamento de Lenguajes y Sistemas Informáticos,
ETS. Ingeniería Informática,
Avd. Reina Mercedes S/N,
Universidad de Sevilla,
Sevilla, Spain
e-mail: {ajvarela,gasca}@us.es

Business Process Management Systems (BPMS). BPMS is a set of software tools to manage business processes.

BPM paradigm follows a life cycle which consists in several stages [17]. During some of these stages different kinds of faults can be introduced:

- In the design stage, business process models could present some design faults (such as deadlock, livelock, starvation, and so on). Design problems are not taken into account in this paper since these are a topic very discussed in various reviews of the literature, [9] [11].
- In the run-time stage, output process faults could be located in the business processes when they obtain unexpected outputs, unexpected messages, unexpected events, or also unexpected performances. Executable business processes usually use external services that are not under our jurisdiction. Thus, it cannot be possible to ensure that the functionality of a certain service changes during the business process life cycle.

Dependability is a significant requirement for many types of companies, for instance: electronic banking and commerce, automated manufacturing, and so on. A failure in their business processes leads them to terrible consequences: lives lost, systems destroyed, security breaches, and so on. Therefore, companies must ensure that their business processes are as dependable as possible before automating them. It must be mandatory to pay attention to the inclusion of measures that allows to reduce the risk of possible faults in business processes from design stages.

In this work, we propose a framework, OPBUS. This framework holds different capabilities but in this paper we only focus on the implementation of the fault tolerance techniques. The main objective of this work is to handle malicious attacks during runtime of business processes. OPBUS enables to execute business processes in an adaptive way by means of handling the security problems using fault tolerance methods. OPBUS is able to modify business processes behavior during runtime according to the actions based on fault-tolerance techniques (i.e., checkpointing and recovery). For example, when a business process is being attacked, OPBUS may determine what activity is under attack and will be able to migrate the running activity another know with well behavior.

This paper is structured as follows: Section 2 presents the framework; in Section 3 an illustrative example is developed; Section 5 shows experimental results that are discussed; in Section 6 related works are discussed, and in the last section, different conclusions and future works are provided.

2 OPBUS Framework Description

OPBUS has been developed based on the main ideas of Model-Driven Development (MDD) [16] and Model-Driven Architecture (MDA) [12]. Model-Driven Development is a software engineering approach where models become key elements in software development. One of the main goals of MDA is to improve the software adaptation to several different technological scenarios, thereby providing a structural separation in the architecture. The solution proposed is an architecture

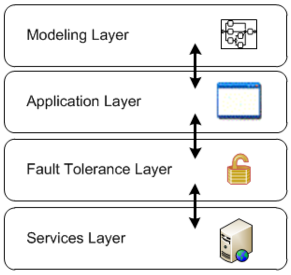

Fig. 1 OPBUS framework

composed of various different levels of modeling. This separation enables the specification of models to a very high level for a particular domain but with non-specific information about the platform where the model will be deployed. MDA introduces the concept of transformation which allows one model to be obtained in one level (target model) from another model or a set of models from another level (source model). OPBUS is structured in several layers as shown in Fig. 1: presentation, application, fault tolerance and services.

2.1 Modeling Layer

This layer is focused on the design of business processes. Business process modeling remains an important factor in the areas of information systems development and business process management. Business process models are defined by graphical models which consist of a set of activity models and execution constraints between them. These models are the main artifact to implement and automatize business processes. The most important and accepted standard is Business Process Modeling Notation (BPMN) by OMG [2]. Although there exist other used methodologies for business process design as Petri Nets, Event-driven Process Chain (EPC), UML Activity Diagram. They are more useful in the academia for research areas but not enough extended in the current industry. Although BPMN is a standard to design, developed models are not executed-oriented. Therefore, a transformation from BPMN models to executable business process is mandatory. Business Process Execution Language (BPEL) [1] is a de facto standard language to implement service-based business processes and lots of commercial tools support it. BPMN could be automatically translated into executable process BPEL [13], and some commercial BPMS make translations from a BPMN diagrams to BPEL processes.

At the moment, in modeling layer, OPBUS provides a tool with BPMN design capabilities. This tool has been equipped with fault diagnosis capabilities using model validation at design time. Likewise, thanks to OPBUS is MDA-based, the developed BPMN models can be very easily transformed into BPEL.

2.2 Application Layer

In this layer, different technologies which are involved in the framework are presented. BPEL environment is necessary to deploy and execute operational business processes. Some of the distinguished commercial tools of BPMS such as Intalio BPM, Borland Together, Oracle BPMS; and other non-commercial such as GlassFishESB, Eclipse BPEL, jBPM, support BPEL. In our proposal, GlassFishESB has been selected. It provides an integrated design environment and an business processes engine for BPEL processes. Likewise, it contains several components to support functions of Enterprise Service Bus (ESB).

Model-based diagnosis has been integrated within the framework. For the diagnosis task, Constraint Programming is applied. Constraint programming is a programming paradigm to solve Constraint Satisfaction Problems (CSPs) which are an extended form to solve problems [14]. ChocoSolver is a CSP solver which provides a constraint programming API to implement constraints models. Although the model could be implemented standalone from other CSP solver tools. ChocoSolver has been integrated within OPBUS. The utility of using CSP is explained in detail in the next sections.

As summarization, application layer is composed of business process engine and a CSP solver. As an assumption, we suppose that both work without faults.

2.3 Fault Tolerance Layer

Although in this section we focus on description of the fault tolerance technique applied in OPBUS, the fault tolerance ideas have been taken into account in every layers. In Fig. 2 different applied mechanism for layers are shown.

OPBUS has been built with a specific fault tolerance layer. This layer is developed with the intention of controlling possible integrity attacks over the tasks involved in the business processes and making the corresponding corrective actions to recover them. In our case, we have applied fault tolerance mechanism focus on the simulation of checkpoint and recovery [10], but in this case oriented to business processes. Although, other fault-tolerant mechanisms have been already equipped successfully, [18].

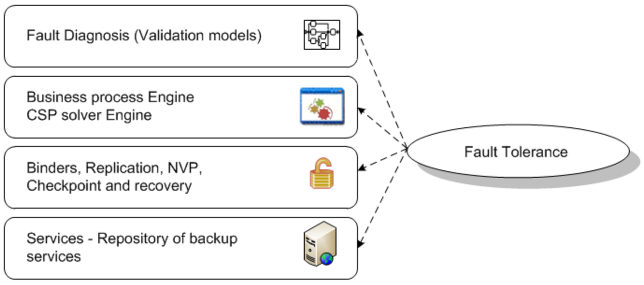
Fault Diagnosis (Validation models)
Business process Engine
CSP solver Engine
Binders, Replication, NVP,
Checkpoint and recovery
Services - Repository of backup
services
Fault Tolerance

Fig. 2 Fault Tolerance within OPBUS

Checkpoint mechanism is based on the idea of saving the state of the system, and in the case of the detection of a fault, recovering the execution of the system in the checkpoint where the stated was saved. We propose to simulate a checkpoint approach for determining faults of integrity in services and only in case of faults to launch any recovery mechanism. The fault tolerance approach mechanism is composed of two parts:

- Integrity sensors (Checkpoints). An integrity sensor has been modeled as a CSP. Sensors takes data information about data inputs and outputs from the services, with those the CSP is formed. The 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 to limit the effects created by a process when faults or errors occur. Compensate handlers allow to rollback the process execution from a specific point, executing a set of tasks to undo the transaction already initiated. We are going to explain compensation handlers in Section 4 in more detail.

3 Illustrative Example

In this section, we present an example of business process, Fig. 3. The business process uses services-based tasks. The process shows a procedure where at least eight services with different functionalities are involved. Process receives a set of data inputs and, at the end, a set of data outputs are achieved. Every service needs a set of inputs and produces a set of outputs. The outputs of some services are the inputs of other services. Thus, some services are dependant each other. Before continuing, it is necessary to take for granted some assumptions in our proposal:

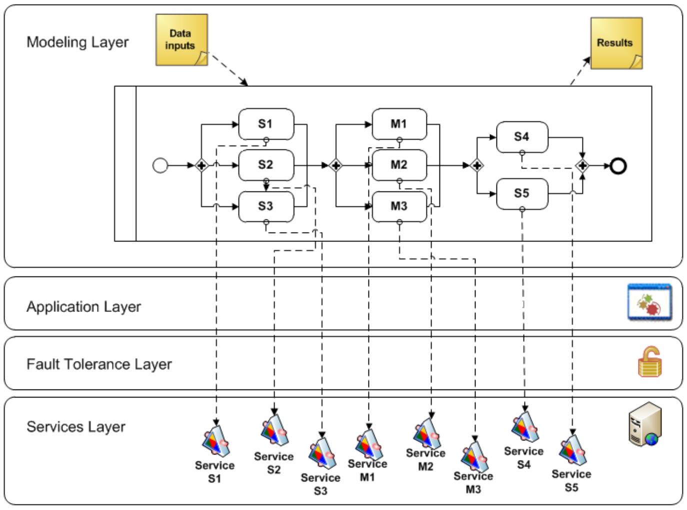

Fig. 3 Example of business process

the business process model has a start event, a single end event and every tasks contribute to finish the process in a correct form, and there do not exist design faults. The network environment where the example has been developed is controlled not to introduce noise in the data. For this reason, we consider that the communications always works without faults or attacks and the data cannot be modified. Therefore, in our example we have simulated that the integrity attacks (faults) can occur in the services.

4 Implementation Details

In this section, the applied solution is described in detail. Some questions has been considered in the solution. One of them is how many integrity sensors to place and where locate them. To find out the optimal number of sensors and where introduce them is a very desirable requirement not to damage or complicate the business process design. This problem has already been studied in the revised literature [5]. In our example, the sensors have been located as shown in Fig. 4. Integrity sensors by means of a CSP resolution return if a service is behaving as it is expected. In a checkpointing approach, the process state will be saved in this checkpoint, but in our approach it is not necessary. Likewise, integrity sensors is able to determine possible fault in a finite set of activities of the business process. In Fig. 4, sensor IS1 covers the services S1,S2 and S3, however IS2 covers the services from IS1 and IS2. To explain the functionality of a compensation handler the next example is used: a bank has a business process which takes the data from the client to do a transaction from the client's account to his credit card to increase the credit of the card. If a service takes a specific client's data and return the amount of an account, but for a fault of integrity this service returns the amount plus other data about another account. This service is not working in compliance with the specification. If this fault is determined, it could be possible to throw an fault exception and, using

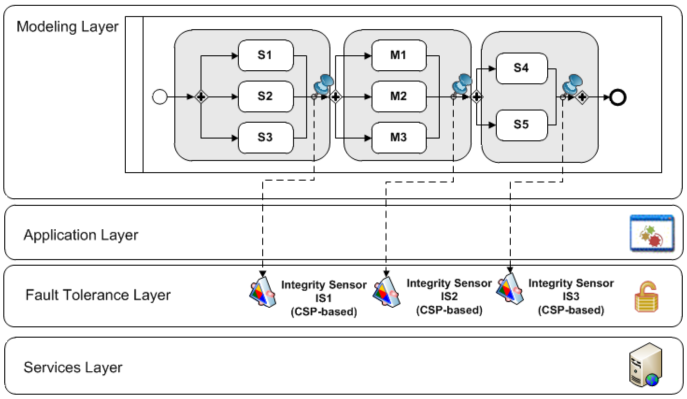
Modeling Layer
S1
S2
S3
M1
M2
M3
S4
S5
Application Layer
Fault Tolerance Layer
Integrity Sensor IS1 (CSP-based)
Integrity Sensor IS2 (CSP-based)
Integrity Sensor IS3 (CSP-based)
Services Layer

Fig. 4 Location of sensors

compensation handlers, undo all transactions carried out from this moment. Compensation handlers can only be launched when an internal business process fault is catched. In described case, the services was working well but the fault came from the functionality of the service.

Compensation is the really useful tool to undo transactions, but it will be applied with another sense. In our case, when an integrity sensor determines any faults in the execution of the business process a fault is launched at the end of the process. After throwing the fault, it is catched by the process and the compensation handler is invoked. Then within of the compensation, the services with failures are re-executed from a backup with correct functionality already tested. This process is shown in the Fig. 5, where the faulty activities has been marked with a red cross within the activities. One consideration have been taken into account, for example if the service M1 has a fault and the service S5 waits any data generated from M1, there is a dependency between M1 and S5, therefore whether M1 has a fault 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.

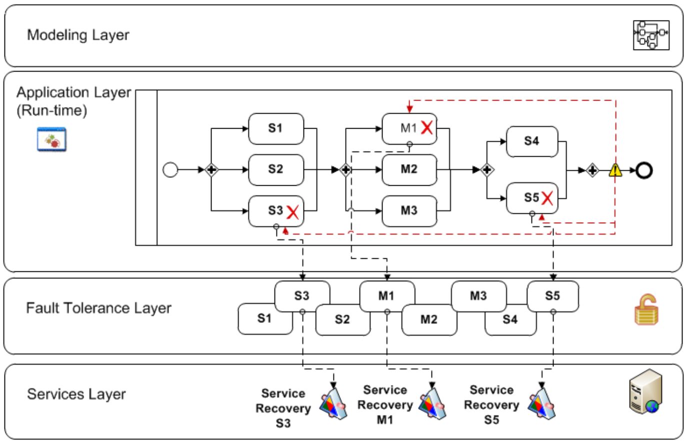

Fig. 5 Recovery business process

5 Performance Evaluation

To test the fault tolerance of the business process, we have simulated the injection of different integrity attack events to trigger the corresponding function on the services. A set of test cases has been executed, they use different number of petitions of the process. The parameter to measure in this case are: time of execution of a process without faults, time of execution of a process with faults, and the number of service with faults. The hardware used to execute the tests is a server Intel Xeon 2.4 GHz, with 8GB RAM and a client Intel Core 2 Duo 2,5GHz with 4 GB RAM. We can

Table 1 Fault tolerance results with 400 petitions to the process

N. fault services	Time with fault [s]	Recovery
1	0.12	100%
2	0.16	100%
3	0.15	100%
4	0.16	100%
5	0.16	100%

highlight that the execution time without fault has obtained the identical values in all cases, about 0.12 seconds, but in case of fault our solution achieves a recovery one hundred percent but at the expense of very low overhead of execution, as shown in Table. 1.

6 Related Work

The dependability has been studied in the context of business process management in [4]. 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 main contribution is the definition of two languages, WSCoL and WSRS, although they are not a supported standard. Likewise, [4] presents some remedial strategies that are mainly focused on the recovery context, but these do not pay attention in the typical solutions in the fault tolerance scope.

In the scope of fault tolerance in BPEL processes there are some contributions, [7][8]. The feasibility of BPEL to implement fault tolerance techniques with BPEL language is studied in [7]. Another work is focused on the dynamic selection of Web Services for the construction of optimal workflows, [8]. 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.

In fault tolerance of distributed systems, checkpointing and rollback recovery approaches are popular [6][10][3]. A well-designed checkpointing algorithm allows to recover a faulty business process from the recently saved state. Other proposals have been developed in other domains such as grid computing [15] and Web Services [4]. Previous works do not consider malicious attacks detection and are focused on finding out what damages of these attacks are.

7 Conclusions and Future Work

This paper presents a framework, OPBUS. Main goals of the framework are: design, implementation, and evaluation of a fault tolerance mechanism in business processes. We show how OPBUS provides different mechanisms, from model validation in modeling layer up to fault tolerance mechanism at run-time. In the fault tolerance topic, OPBUS control business process detecting possible integrity faults in services and it makes corresponding actions according to the integrity attacks. The experimental results show how the fault-tolerant mechanism handle the integrity faults efficiently with low overheads and with a recovery index of the one hundred percent. In the near future, we will add new policies to further improve performance, for example, by adjusting sensors on-line according to different security situations.

Acknowledgements. This work has been partially funded by Consejería de Economía, Innovación y Ciencia of Regional Government of Andalusia project under grant P08-TIC-04095, and by Spanish Ministerio de Ciencia e Innovación project under grant TIN2009-13714, and by FEDER (under ERDF Program).

References

1. Business process execution language (2008), `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
2. Business process model and notation (2009), `http://www.omg.org/spec/BPMN/1.2`
3. Baldoni, R.: A communication-induced checkpointing protocol that ensures rollback-dependency trackability. In: FTCS 1997: Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS 1997), Washington, DC, USA, p. 68. IEEE Computer Society, Los Alamitos (1997) ISBN 0-8186-7831-3
4. Baresi, L., Guinea, S., Plebani, M.: Business process monitoring for dependability. In: WADS, pp. 337–361 (2006)
5. Borrego, D., Gómez-López, M.T., Gasca, R.M., Ceballos, R.: Determination of an optimal test points allocation for business process analysis. In: IEEE/IFIP Network Operations and Management Symposium Workshops, BDIM 2010 (2010) ISBN 978-1-4244-6039-7
6. Cao, G., Singhal, M.: Checkpointing with mutable checkpoints. Theor. Comput. Sci. 290(2), 1127–1148 (2003)
7. Dobson, G.: Using ws-bpel to implement software fault tolerance for web services. In: EUROMICRO-SEAA, pp. 126–133 (2006)
8. Huang, L., Walker, D.W., Rana, O.F., Huang, Y.: Dynamic workflow management using performance data. In: IEEE International Symposium on Cluster, Cloud, and Grid Computing, pp. 154–157 (2006)
9. Huang, S.-M., Chu, Y.-T., Li, S.-H., Yen, D.C.: Enhancing conflict detecting mechanism for web services composition: A business process flow model transformation approach. Inf. Softw. Technol. 50(11), 1069–1087 (2008)
10. Kim, J.L., Park, T.: An efficient protocol for checkpointing recovery in distributed systems. IEEE Trans. Parallel Distrib. Syst. 4(8), 955–960 (1993)
11. Mendling, J., Moser, M., Neumann, G., Verbeek, H.M.W., Vandongen, B.F.: Faulty epcs in the sap reference model. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 451–457. Springer, Heidelberg (2006)
12. Miller, J., Mukerji, J.: Mda guide version 1.0.1. Technical report, Object Management Group, OMG (2003)

13. Ouyang, C., van der Alast, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Translating bpmn to bpel (2006)
14. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
15. Shi, X., Pazat, J.-L., Rodriguez, E., Jin, H., Jiang, H.: Adapting grid applications to safety using fault-tolerant methods: Design, implementation and evaluations. Future Generation Computer Systems 26(2), 236–244 (2010)
16. Stahl, T., Völter, M.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, Chichester (2006) ISBN 978-0-470-02570-3
17. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business Process Management: A Survey, pp. 1–12 (2003)
18. Varela-Vaca, A.J., Gasca, R.M., Borrego, D., Pozo, S.: Towards dependable business processes with fault-tolerance approach. In: 3rd International Conference on Dependability, DEPEND (2010)
19. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2007) ISBN 978-3-540-73521-2