Supporting SOA Resilience in Virtual Enterprises

Roque O. Bezerra[1(✉)], Ricardo J. Rabelo[1], and Maiara H. Cancian[2]

[1] Department of Automation and Systems Engineering,
Federal University of Santa Catarina, Florianopolis, SC, Brazil
{roque.bezera, ricardo.rabelo}@ufsc.br
[2] Estácio Florianopolis, Rodovia SC401 Km 01, Florianopolis, SC, Brazil
maiara.cancian@estacio.br

Abstract. Computing systems are essential nowadays for the execution of companies' business processes and should keep operating permanently. Modern approaches, as Service Oriented Architecture (SOA), have been gradually adopted by companies to implement their systems. This paper exploits a Virtual Enterprise (VE) scenario where its members' systems are available as services and are selected to support the VE operation itself. Regarding VE properties and inspired in the autonomic computing paradigm, a resilience architecture and system have been designed and implemented to help VE's supporting system to recover from services' faults, respecting the business processes' QoS in place. Results are presented and discussed in the end.

Keywords: Reference business processes · Service Oriented Architecture
Resilience · Fault tolerance · Virtual Enterprises

1 Introduction

Computing systems have become essential for the execution of companies' business processes. As such, keep them permanently operating is one of their major concerns [1]. SOA (Service Oriented Architecture) has been increasingly adopted by SMEs to foster newer business models, based on larger scale provision and offering of software services that are distributed over the Internet and that can be accessed on demand, from everywhere, anytime from pervasive providers from digital ecosystems [2].

This paper deals with Virtual Enterprises (VE). One of VE's properties refers that its members share resources and working principles as well as they have enough IT preparedness to participate in VEs. In this sense, this work exploits the scenario where VE members can share their software services assets, and a SOA/services-based and cross-boundary system is temporarily and dynamically created to support the execution of the VE's business processes throughout the VE life cycle [3].

This created system is therefore composed of (loose-coupling) services from the currently VE members and, eventually, also from their IT supporting business partners, creating a large-scale distributed system. In this scenario, several faults (e.g. services unavailability) can happen during the system execution so causing problems in the VE and related businesses if proper measures are not applied [4].

L. M. Camarinha-Matos et al. (Eds.): PRO-VE 2018, IFIP AICT 534, pp. 111–123, 2018.
https://doi.org/10.1007/978-3-319-99127-6_10

This paper addresses this issue from the IT resilience perspective. IT resilience generally refers to guaranteeing the operation of the system under control and its recovery in the presence of faults or high degradation within acceptable costs [5].

VEs bring up additional resilience requirements and research opportunities regarding their collaborative, dynamic and open natures. However, it hasn't been tackled much in specific in the literature. Despite the complexity of the problem, most of the evaluated works on SOA resilience doesn´t consider much the intrinsic VE dynamics in terms of members composition, and they usually assume a too simplistic IT reality of SOA problems when applied in real business cases [1, 6].

This paper presents a resilience architecture and supporting system to deal with the VE scenario where a heterogeneous SOA-based system should remain operating when the involved services have problems or the VE's composition change (hence the respective services should be replaced as well). It is an ongoing work and has been developed under the action-research methodology.

This article is organized as following. Section 1 has introduced the problem and the objectives of the work. Section 2 summarizes the requirements of a resilience architecture for SOA in VEs. Section 3 presents the proposed architecture. Section 4 describes the prototype and the achieved results. Section 5 presents some preliminary conclusions and the next main steps of this work.

2 Literature Review and VE Resilience Aspects

Resilience and fault tolerance terms are sometimes used as synonyms or just used differently depending on the scientific 'community'. Some authors, like [7, 8], take resilience as a wider perspective for systems reliability. They consider that fault tolerance term is more appropriate to be used in the cases where faults treatment is handled at system's design, while resilience would be more suitable in more flexible, open, dynamic, evolving and less prescriptive architectures, which is the case of the proposed work in this paper. Adaptive fault tolerance is another term found out in the literature and it seems equivalent to resilience [7].

After an extensive review on theoretical foundations of collaborative networks and VEs as well as of computing resilience and fault tolerance, a number of aspects were identified as important to be taken into account when a resilience architecture/system is going to be developed for dealing with VE.

A systematic literature review (SLR) [9] was done upon five international scientific repositories looking for works combining the areas of SOA/services, resilience/fault tolerance and VE (and its other equivalent terms). Almost three thousand papers were found out in the search, and 27 were preliminary selected. Due to space limitation to mention them in the references, only the nine taken as the most relevant ones (for the purpose of this work) are listed.

Table 1 summarizes these works against the identified VE-related aspects as well as highlights the envisaged contribution of this work. By "supported" it is meant at least *some* level of guarantee. By "not (yet) supported" it is meant that the given feature is not currently supported but it is planned to be in the next version of the work. By "not supported" that the given feature is not anyhow supported in this proposal.

Table 1. Summary of the literature review

VE-related aspect	Works [10–18]	Proposed work
1. Newcomers can join a VE and current members can leave it during its execution. This means that the composed SOA/services-based supporting system should be also dynamically recomposed accordingly in order to keep the VE operating gracefully	−Most of them support dynamic discovery and re-composition	Supported
	−These works only consider previously defined alternative services, and a discovery and execution in local & intra-enterprise environments	Supported
	−None works consider the dynamics and scalability of members' composition	Supported
	−Only one work considers the dynamics and scalability of existing services provision	Supported
2. VE members are independent companies and usually adopt different models in their BP modeling, impacting the services' functional requirements and the way SOA layer interacts with the Business and Infrastructure layers This relates to very well defined scope of responsibilities (and hence the actions) of each layer	−Only one work offers some level of integration with the Business layer	Supported
	−All works support integration with the Infrastructure layer	Supported
	−Only four works adopt standard BP models, but none of them make use of this to facilitate services discovery and interop	Supported
3. VE members implement their services in different technologies, IT standards & patterns, security mechanisms, granularities, deployed in different servers, and registered using different signatures and repositories. The effective system (re)composition and execution require syntactic or semantic interoperability mediation	−Most of works assume a homogeneous environment, basically composed of web services, XML and SOAP	Not (yet) supported
	−Only three works use some mediation (via ESB [*Enterprise Service Bus*]) to support larger interoperability	Supported
	−Only two works offer some security support	Not Supported
4. Companies (and so their services) can be linked to several VEs simultaneously in their different stages, meaning that a given service can have several instances/tenants in execution responding to different VEs' QoS metrics and multiple SLAs	−Most of works deal with end-to-end QoS	Supported
	−Three works also monitor and handle temporal restrictions of the individual services	supported
	−None works support multi-tenancy	Not supported
5. Companies' services are in theory permanently running. However, services can become unavailable or fail during different VE phases: before getting bound to given VEs; before being invoked by a given VE's business processes (BP); and during their execution in given VEs	−All works checks services only in the execution phase	Supported

(*continued*)

Table 1. (*continued*)

VE-related aspect	Works [10–18]	Proposed work
6. Each business a VE is related to can have different priorities (i.e. weights) in terms of the most critical QoS metrics to pursuit, impacting the general costs and hence the feasibility of the resilience policy in place	−Only four works support some level of parametrization, weighting or prioritization	Supported
7. The access to VE members' services should follow the VE governance model and possibly the gov model of the long-term alliance the members belong to	−Not supported	Not supported
8. The general computing infrastructure to support the execution of the VE resilience system should ideally be deployed in servers that are not dependent of any given VE member as it can leave the VE anytime. The own resilience system should ideally be resilient in order to mitigate its complete fail in the case of problems	−All works have developed centralized architectures/systems to handle resilience	Supported
	−All works do not support self-resilience	Not (yet) supported
9. The VE's legal obligations only end after all the contracted aspects have been fulfilled. This means that the VE members' services should be kept 'connected' even after the final 'product' (which originated the creation of a VE) has been delivered	−Not supported	Not supported
10. The replacement of a given VE member (and its services) by another members (and its services) should consider implementation issues, like the service's components lock-in and business duties. Yet, the creation of services' replicas may imply replicating other components as well (e.g. a database)	−Not supported.	Not (yet) supported
11. VE members are dealing with real businesses, carried out collaboratively and in a distributed way. Resilience supporting systems should ideally act pro-actively close to the involved members' services in order to prevent the VE from generally failing because given members' services have got down	−Half of works do it pro-actively	Supported

3 The Proposed SOA-Based Resilience Architecture and System

This section presents the proposed resilience architecture for VEs.

Considering (i) this is an ongoing work; (ii) the complexity of some aspects related to VE resilience (Table 1); and (iii) that some of them involve issues that are not even totally or well solved in the distributed systems area; the proposed architecture showed in this paper has been designed to handle (at different levels of depth) only the aspects pointed out in Table 1.

3.1 The Resilience Architecture's Rationale

A number of general design principles have been considered in the architecture:

(a) All VE members belong to a long-term collaborative alliance of type VBE (*Virtual organization Breeding Environment*), which is grounded on trust and members' autonomy, and whose members intrinsically have the willingness to collaborate and share resources [4]. This means that, as a general rule, their services can be made available to be accessed by other VBE members and hence VEs [6]. This assumption relies on the fact that any VBE member must respect a number of common principles of work and introduce them in their companies so that their general differences get hidden to other companies. This all refers to the so called members' preparedness [4].

(b) IT preparedness is one of the pre-conditions for a company to be member of a VBE and VE [4]. This means that their services and computing infrastructure should be previously prepared (at several levels) and duly wrapped so as to be also used by other client systems, including the resilience system. This, however, does not mean forcing all companies to adopt the same IT or standards, although this actually happens in plenty of cases in real life SMEs when integrating with larger enterprises.

(c) When a given company leaves a VE its services keep available to be accessed by the resilience system so as to replace problematic services, regarding the collaborative nature of VBEs (as in [6]). However, their effective use depends on BP's activities restrictions and/or technological factors;

(d) The architecture separates the whole resilience actions into three inter-dependent actors ('responsibility' layers, as in [18]), relating to the model, runtime and deployment views: (1) the planning layer (as a BPM-like environment, where business processes and activities are defined and services are discovered and bound to); (2) the SOA/services layer (where services are deployed, made available, also discovered and executed); and (3) the middleware/operating system layer (responsible to support the execution and communication of all services – and of the own resilience system). A number of assumptions are taken within each layer. One of the most important ones is the adoption of a given reference model for BP modeling (as a VBE/VE common neutral model for internal interoperation), based on which all VBE members would have their services developed

according to, although implemented in different technologies, semantics and granularities, as in [19];

(e) The resilience actions can require human intervention (e.g. QoS relaxation) in the case of unsolvable situations so as to keep the VE operating (as in [6]);

(f) The resilience architecture/system has to be decentralized and distributed so as to prevent central points of faults, as in [8];

(g) Each new system's reconfiguration/re-composition has a cost, which should be measured and evaluated before being set up, as in [16].

In this current version of the prototype only services' *faults* are treated. No services' *degradation* analysis is performed.

Five types of general *faults* can be treated by a resilience system for SOA-based systems [14]: *publication, discovery, composition, binding,* and *execution*. A number of very concrete faults are associated to each one. *Publication* and *Binding* faults are actually not considered as necessary to be treated in the proposed architecture. It is assumed that *publication*-related faults (e.g. *wrong publication, wrong interface* and *lookup faults*) are resolved at VBE level when every company properly register its services in their repositories and make them available in the VBE's services federation. In terms of *binding*, it is assumed that the related faults (*wrong binding, binding denied* and *access denied*) are resolved at BPM level when the BPMN and BPEL process are generated, and that services are 'naturally' available as their owners belong to the given VE and VBE (or to their IT business partners).

In terms of *Discovery* and *Composition*, the faults to be treated by the resilience system [14] refer to *services inexistence, services unavailability*, *services inadequacy* (e.g. inadequate QoS) and *discovery fail*. This can happen when services are being bound to BPs' activities. In terms of *Execution* faults, these three faults can also happen when services are going to be *invoked by the BPEL* process and when *services get crashed*. Yet, as the resilience system also monitors the BPEL process another possible fault is the *BPEL process crash*.

Two other SOA-related faults [14] are treated in other layers: the *reserved communication port* fault, which is not up to the SOA layer/resilience system to solve as ports are automatically defined by the infrastructure and middleware layer/systems. The *incorrect result* fault is resolved by the high-level applications, which understand the business logic associated to the BPs' activities.

3.2 The Architecture

The architecture has been devised to support both VE-related aspects (Table 1) and the core design principles, previously described. In order to facilitate its explanation and due to space restrictions, only a general description will be provided, besides mixing some general aspects of implementation and execution.

IT resilience can be addressed from different approaches. Regarding that the desired resilience architecture should monitor its own state and adapt itself in the presence of faults, the autonomic computing approach has been chosen. The *self-inspection* and *self-adaptation* techniques [20] are used to implement that.

The architecture is designed to cope with the discovery, composition and execution faults applying two approaches: *services replaceability* (replacement of the faulty service by an equivalent one [10]) and *services provision and migration* (creation of replicas or dynamic migration of the faulty service to other servers [20]).

The architecture is organized into three layers regarding their role (Fig. 1). Their components work in two different moments: when the VE is being created (Project Phase) and when the supporting service-based system is composed and set up; and when it is executed (Execution Phase). The resilience system will then take care of this VE system.

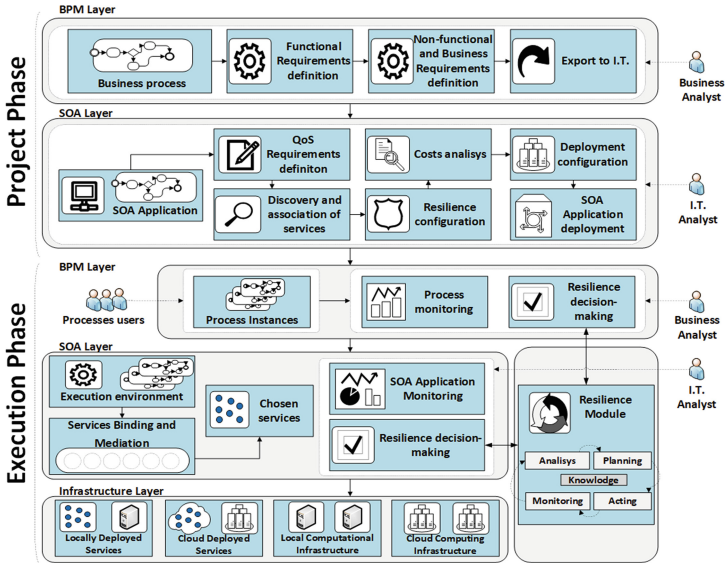

Fig. 1. The proposed architecture and resilience phases

Project Phase:

A VE is created after some steps [4], including the one where the so-called VE's coordinator indicates, in the planning/BPM layer, the actual BPs each VE member will be responsible for. In the implemented prototype, the open standard *UBL* BP Reference Model [21] has been adopted regarding its target on supply chains. Each of its 68 BPs has a number of pre-defined activities and documents to be exchanged between members. As services are dynamically discovered and bound to the involved BPs' activities, only a functional reference of the required services are provided, as in [22]. This provides higher flexibility to the resilience system in its search for equivalent services when it decides to replace the unavailable ones. The VE coordinator's business analyst also specifies the non-functional requirements (as QoS attributes) for each BP's activity, which are used by the resilience system both to control the time constraints of each BP's activity and the BP's end-to-end QoS, as in [6]. All this has been developed using the *Eclipse* and *IBM Websphere*'s APIs.

Once the VE's services are finally discovered and bound, a BPEL (*BP Execution Language*) file is generated as the result of the VE planning, previously modeled (in BPMN). This BPEL is deployed as a SOA application and put into the execution environment (a BPEL engine based on the *Apache ODE*). This is done interacting with the supporting infrastructure (via a *REST API* and *Docker*) and deploying the required services: the execution environment, the involved VE members' services and the own resilience services-based system. Each deployed service may have different numbers of replicas (not implemented yet in this prototype). The replication level and deployment policies can be defined by IT analysts, meaning that the resilience "level" of the own resilience system can be configured.

Execution Phase:
In this phase the services-based system generated to support the VE's BP execution starts to run in the sense that the involved VE members' services are invoked by the BPEL process. The resilience system supervises the BPEL process (in the SOA layer, mainly for trying to guarantee the end-to-end QoS) as well as the involved services (in the infrastructure layer, for trying to keep up the VE operating).

The resilience system does not take care of its own resilience at all. Its modules are implemented as *threads* and communicate with each other using synchronized queues coded in Java. The communication with the other modules uses the SOAP protocol (*point-to-point*) and the *Apache ActiveMQ* (*JMS publish-subscribe* communication middleware). However, the BPEL engine is also replicated and its state is synchronized (using the *Infinispan* framework). This is part of the self-resilience strategy of the model as another replica can assume the execution of the process in the case a given one gets unavailable, making the resilience system more reliable.

The resilience system has been designed to perform the following main activities, which are based on the *MAPE-K* [23] reference autonomic computing model: it *m*onitors the "system" (the BPEL process and services); it *a*nalyses the system entities' status; it *p*lans actions in the case of services faults; and it *e*xecutes actions to keep the VE operating. The *k*nowledge part (to be used by the resilience system to evolve) is not supported in the current prototype.

Services availability is monitored using *endpoint monitoring* via heartbeat requests/ "ping" [24], using two techniques: a *fail fast* every 500 ms to check if services are listening to their ports and, complementarily, a *timeout* of 1000 ms [25].

The resilience system performs an 'expansion cycle' of services discovery (and further composition and binding) during its execution in the case a given service has a problem: it starts by searching for a new service in the respective VE member's repository. If no equivalent service is discovered then the search is expanded to the other VE's members. It ends with a wider search in the VBE's and IT business partners' repositories (i.e. the services replaceability approach). In the case no services are found out then the system will try to deploy the faulty service in another (previously defined) computing infrastructure (i.e. the services provision approach).

An ESB (*Camel ESB*) is used as a complementary entity to help in the services binding and mediation avoiding point-to-point and tight-coupling communication. It pro-actively checks if the invoked services keep being available and responding to the required QoS, and sends this information to the monitoring module. The ESB is kept

permanently updated when new services are bound to given BPs or when members composition change. Besides that, in spite of the implemented prototype has only considered WS-* services, the use of the ESB allows supporting "any" other services' implementation technologies.

4 Results

This section presents the VE scenario and computing prototype implemented in a controlled environment to quantitatively assess the proposed resilience architecture.

This scenario refers to a hypothetic customer who asks for a given product close to a given VBE's company. This product is basically composed of three parts, being one part produced by this company. This company then triggers the process of VE creation, ending up by forming the following VE: this company ('Partner 1') is the *VE Coordinator* and interacts with the customer; and 'Partner 2' and 'Partner 3', which manufacture the other two product's parts. These two partners should send their parts directly to Partner 1 once they have been finished for the final product assembly.

This VE's plan is showed in the Fig. 2, which is the graphical representation of the BPEL file generated from the respective BPM/BPMN modeling. This plan reflects the standard flow of activities specified in the UBL process '*Ordering Process'*.

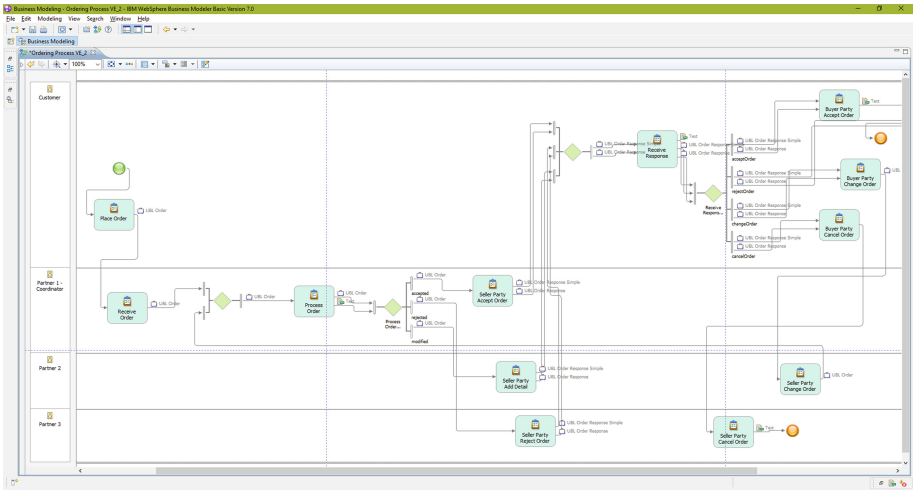

Fig. 2. The business process scenario and related BPEL file

It was simulated a scenario where 100 customer orders arrives to the VE and then the resilience system should try to keep the supporting VE system running and attending the BP's end-to-end QoS in the presence of several services' faults.

Due to space restrictions only the *discovery* and *execution* faults will be showed in this section, i.e. it is assumed that *binding* faults (in the BPEL file) will not happen, although being supported by the developed resilience system.

A number of performance indicators would have to be used to measure the many aspects of resilience in the implemented model. Considering the goals of the current stage of the work, two reference performance indicators [26] are so far used to measure the "quality" of the resilience system: the *"resilience time"* (the time spent to recover from a local fault without violating the global end-to-end QoS, including all the communication, latency and processing times – *'reaction time'* in Fig. 3, line "="); and *"End-to-End violation"* (the number of times and process' instances the end-to-end QoS has been violated - *'process instance duration'* in Fig. 3, line "-").

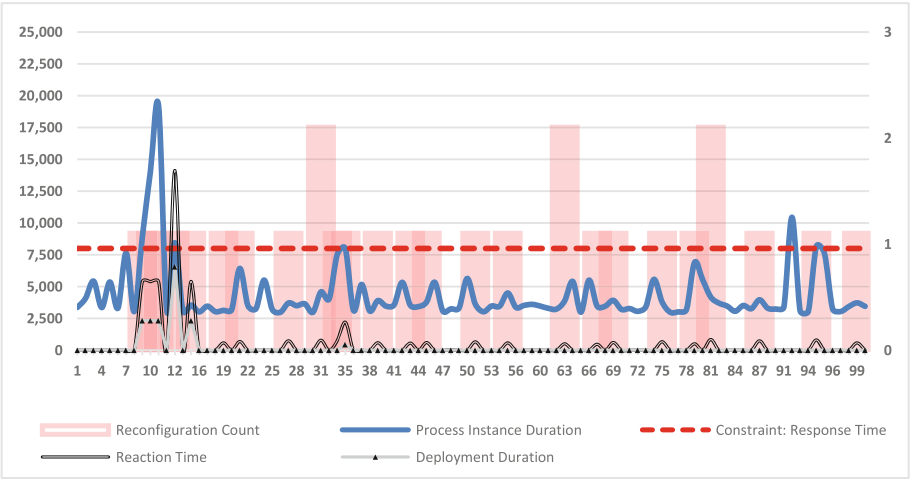

Fig. 3. Experimental results

In this sense, there are three possible ending situations for each VE's customer order: (i) the VE has operated without any end-to-end QoS violation; (ii) the VE has operated within an acceptable number (to be determined by the VBE board or the VE members) of end-to-end QoS violations; and (iii) the VE could not operate as properly due to severe problems in its supporting services.

These situations are actually related to the VE-related aspect 11 (Sect. 2), as the ultimate goal of the resilience system is to "guarantee" that the VE keeps operating and respecting the required end-to-end QoS.

In the implemented scenario each VE member has five available services functionally equivalent to execute each of the standard BP's activities. This means that any of them can be used by the resilience system to automatically replace a faulty service. The services unavailability is randomly set up in the prototype. The replacement strategy is performed via the 'expansion cycle', as previously explained.

The end-to-end QoS value for the UBL process 'Ordering Process' was set up as 8 s (line "- -" in Fig. 3), so it should be observed by the resilience system when summing the individual response times of all the involved services (line "="). The *y* axis represents the time, whereas the *x* axis represents the VE' instance. *Y* axis is also

used to represent the number of services replacements (shaded bars - *'reconfiguration count'*) necessary to recover from services unavailability in the 100 VE's instances.

One thousand services were deployed in the so-called VBE's services federation, being their signatures and QoS attributes also randomly generated when registered. Their response time was randomly assigned with a sleeping time when services are invoked. The computing infrastructure was deployed in three distributed Intel servers in a link of 100 Mbps and a mean latency time of 2 ms.

The resilience system starts monitoring the involved services as soon as the VE is created and services are bound to its BPs. In this experiment, the (simulated) faults start to happen from instance 7 on, when services get unavailable. The resilience system then performs its 'expansion cycle'. The shaded bars indicate the number of services replacement per BP's composition per VE's instance.

The process' instance duration (line "▬") is quite variable. This is due both to the natural variability of services availability/re-composition time and to the variable execution flows of the BP activities (Fig. 2).

The 'deployment duration' (line "▲") shows when the services provision strategy steps in after all the attempts to search for a substitute service in the 'expansion cycle' have failed (*discovery fault*), which took 6.000 ms. This provision took about 2.500 ms to be executed. However, it is necessary to wait for the services initiation in the server and for the re-composition completion. Summing all the other related actions this ended up taking about 20.000 ms, which violates the required global QoS.

As a final average, the resilience system could maintain the VE operating well in 93% of its instances, which can be considered as quite acceptable in general terms.

In terms of computational complexity, the main variables involved in the system execution (e.g. number of services and number of instances) had a *linear* complexity, which seems to be also quite reasonable for a VE scenario.

This prototype and achieved results tried to show at which extent the VE-related aspects 1-2-3-4-5-6-8-11 listed in Table 1 (Sect. 2) were somehow supported.

5 Final Considerations

This paper has presented preliminary results of an ongoing research that, at last, aims at conceiving and implementing an autonomic resilience architecture and system to sustain the Virtual Enterprise (VE) operation in the presence of diverse faults in its SOA/services-based supporting systems.

A core assumption of this architecture is that VE members all belong to a long-term business alliance grounded on trust, collaboration and resources sharing, including software assets. IT preparedness is one of the pre-conditions for a company to be member of a VBE and VE, being its systems properly wrapped as services. Therefore, the VE supporting system is formed by a composition of members' services involved in the diverse business processes related to the given VE' business and it is the one that the resilience system supervises.

VE brings up a number of particular requirements to be supported when compared to a resilience system for "any" distributed system. The devised architecture has been designed to cope with most of them although adopting some assumptions. Issues that

are not supported by the architecture and system were also identified, to be highlighted security, governance and the *Knowledge* part of the MAPE-*K* model.

A system prototype was implemented and assessed using open source tools and IT standards, in a controlled environment. Three types of services faults were tested. In general, it can be said that the system showed a good potential to support resilience actions within feasible costs as well as in terms of computing complexity.

The mean result of 93% of recovery could be actually better. The simulated environment has actually "forced" each service to get unavailable many times during the VE's instances execution, which is not realistic in minimally robust infrastructures. This number also shows the importance of a resilience system for VEs. Almost all VE instances would have had problems to keep operating within the required QoS level if the diverse detected services' faults were not properly treated.

Next main steps of this research include: the consideration of services *degradation faults*; a self-resilience model; the implementation of services in multiple technologies and protocols; and a support for stateful services' replicas, which brings tough issues to be coped with in terms of services coordination and of evolution of system's state.

References

1. Rabelo, R.J., Baldo, F., Alves-Junior, O.C., Dihlmann, C.: Virtual enterprises: strengthening SMES competitiveness via flexible businesses alliances. In: North, K., Varvakis, G. (eds.) Competitive Strategies for Small and Medium Enterprises, pp. 255–272. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27303-7_18
2. Brzeziński, J., et al.: Dependability infrastructure for SOA applications. In: Ambroszkiewicz, S., Brzeziński, J., Cellary, W., Grzech, A., Zieliński, K. (eds.) Advanced SOA Tools and Applications. Studies in Computational Intelligence, vol. 499, pp. 203–260. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-38957-3_5
3. Bezerra, R.O., Cancian, M.H., Rabelo, R.J.: Enhancing network collaboration in SOA services composition via standard business processes catalogues. In: Camarinha-Matos, L. M., Afsarmanesh, H., Fornasiero, R. (eds.) PRO-VE 2017. IAICT, vol. 506, pp. 421–431. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65151-4_38
4. Afsarmanesh, H., Camarinha-Matos, L.M., Ermilova, E.: VBE Reference Framework. In: Camarinha-Matos, L.M., Afsarmanesh, H., Ollus, M. (eds.) Methods and Tools for Collaborative Networked Organizations, pp. 35–68. Springer, Boston (2008). https://doi.org/10.1007/978-0-387-79424-2_2
5. Annarelli, A., Nonino, F.: Strategic and operational management of organizational resilience: current state of research and future directions. Omega **62**, 1–18 (2015)
6. Vernadat, F.B.: Technical, semantic and organizational issues of enterprise interoperability and networking. IFAC Proc. **42**(4), 728–733 (2009)
7. Strigini, L.: Fault tolerance and resilience: meanings, measures and assessment. In: Wolter, K., Avritzer, A., Vieira, M., van Moorsel, A. (eds.) Resilience Assessment and Evaluation of Computing Systems, pp. 3–24. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29032-9_1
8. Banatre, M., Pataricza, A., Moorsel, A., Palanque, P., Strigini, L.: From Resilience-Building to Resilience-Scaling Technologies: Directions - ReSIST NoE Deliverable D13. Technical reports (2007). http://hdl.handle.net/10451/14107

9. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical report EBSE-2007-01 (2007). https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf
10. Cardellini, V., et al.: MOSES: a framework for QoS driven runtime adaptation of service-oriented systems. Softw. Eng. **38**(5), 1138–1159 (2012)
11. He, Q., et al.: Localizing runtime anomalies in service-oriented systems. IEEE Trans. Serv. Comput. **10**(1), 94–106 (2017)
12. Weidong, W., Liqiang, W., Wei, L.: A resilient framework for fault handling in web service oriented systems. In: IEEE International Conference Web Services (2015)
13. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimization in service-based systems. IEEE Trans. Softw. Eng. **37**(3), 387–409 (2011)
14. Ardagna, D., Baresi, L., Comai, S., Comuzzi, M.: A service-based framework for flexible business processes. IEEE Softw. **28**(2), 61–67 (2011)
15. Menascé, D., Gomaa, H., Malek, S., Sousa, J.: SASSY: a framework for self-architecting service-oriented systems. IEEE Softw. **28**(6), 78–85 (2011)
16. Stein, S., Payne, T.R., Jennings, N.R.: Robust execution of service workflows using redundancy and advance reservations. IEEE Trans. Serv. Comput. **4**(2), 125–139 (2011)
17. Hummer, W., Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: VRESCo – vienna runtime environment for service-oriented computing. Service Engineering, pp. 299–324. Springer, Vienna (2011). https://doi.org/10.1007/978-3-7091-0415-6_11
18. Friedrich, G., Fugini, M.G., Mussi, E.: Exception handling for repair in service-based processes. IEEE Trans. Softw. Eng. **36**(2), 198–215 (2010)
19. Schratzenstaller, W.M.K., Baldo, F., Rabelo, R.J.: Semantic integration via enterprise service bus in virtual organization breeding environments. In: Nguyen, N.T., Trawiński, B., Fujita, H., Hong, T.-P. (eds.) ACIIDS 2016. LNCS (LNAI), vol. 9622, pp. 544–553. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49390-8_53
20. Pääkkönen, P., Pakkala, D.: Mechanism and architecture for the migration of service implementation during traffic peaks. Serv. Oriented Comput. Appl. **9**(2), 193–209 (2015)
21. OASIS.: Universal Business Language Version (UBL) 2.1. 2013 (2016). http://docs.oasis-open.org/ubl/UBL-2.1.html
22. de Souza, A.P., Rabelo, R.J.: A dynamic services discovery model for better leveraging BPM and SOA integration. Int. J. Inf. Syst. Serv. Sect. (IJISSS) **7**(1), 1–21 (2015)
23. Kephart, J.O., Chess, D.: Dm.: the vision of autonomic computing. Computer **36**(1), 41–50 (2003)
24. Homer, A., Sharp, J., Brader, L., Narumoto, M.: Cloud design patterns: prescriptive architecture guidance for cloud applications. Microsoft Pattern Pract. 238 (2014)
25. Nygard, M.T.: Release it!: design and deploy production-ready software. In: Raleigh, N.C. (ed.) Pragmatic Bookshelf (2007)
26. Liu, D., Deters, R., Zhang, W.J.: Architectural design for resilience. Enterp. Inf. Syst. **4**(2), 137–152 (2010)