On the Design of a Domain Specific Language for Enterprise Application Integration Solutions *

Rafael Z. Frantz[1], Carlos Molina-Jimenez[2], Rafael Corchuelo[3]

[1] UNIJUÍ University, Department of Technology, Ijuí, Brazil
rzfrantz@unijui.edu.br
[2] School of Computing Science, University of Newcastle, UK
carlos.molina@ncl.ac.uk
[3] Universidad de Sevilla, ETSI Informática - Avda. Reina Mercedes, s/n. Sevilla 41012 Spain
corchu@us.es

Abstract. Enterprise application integrations involve the participation of several existing applications with which the integration solution exchanges data over LANs and the Internet. In these scenarios, operations might occasionally produce exceptional results at runtime due to impairments introduced by the electronic infrastructure such as node crashes, messages lost, delayed or incorrectly composed by applications. To address the problem, the paper suggests a domain specific language to specify the integration solution: it produces platform-independent models and has built-in primitives to produce events that notify of potential exceptional situations. The paper also shows how these events can be processed by an event condition action–based monitor to trigger recovery actions.

Key words: Enterprise Application Integration, Domain Specific Language.

1 Introduction

The computer infrastructure of a typical today's enterprise can be conceived as an heterogenous set of applications (termed the software ecosystem) that includes tens of applications purchased from different providers or built at home. An application is a piece of software that performs an independent and specific business function. Examples of typical functions are calculation of salaries, tax liability, etc. A recurrent challenge that appears in enterprises is the need to enhance the functionality of their software ecosystem by making some of the existing applications to interoperate with others. In the literature, this problem is known as Enterprise Application Integration (EAI) and is all about making two or more existing applications, that belong to the same

* The first author conducted part of this work at the University of Newcastle, UK as visiting member of staff. His work is partially funded by the Evangelischer Entwicklungsdienst e.V. (EED). The second author is partially funded by UK EPSRC Platform Grant No. EP/D037743/1. The third and first authors are partially funded by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, and TIN2008-04718-E).

enterprise, to synchronize their data or to create new functionalities on top of them; in either case, the software that implements the integration is called the EAI solution.

The integration of two or more applications inevitably involves access to each other's data. In the simplest case, an application integration would involve the transmission of a single unit of data (for example, via a RPC) directly from an application to another. However, in practice, application integration normally involves a large number of interactions among applications that, in general, result in a rather complex flow. In these scenarios, the data normally endures some processing between the source and the destination and is subjected to several constraints such as order, timing and validation. For example, data from two or more applications can be validated against syntax errors, then merged and re-formatted to compose a single message, and then delivered to a third application before the expiration of a deadline. To handle this complexity, it is convenient to regard the EAI solution, as a business process that interacts with the participating applications (in this article we call them assets).

From these observations, it follows that the final job of the designer is to produce the EAI solution that implements the new business process on top of the existing computer infrastructure. Two factors make this problem a challenging task: First, computer technology is not static but constantly changing. For example, a piece of software or hardware can be upgraded. A good solution should be able to cope with computer infrastructure evolution, without drastic re-designs. Second, computer infrastructure is far from being 100% reliable. For example, a communication network can fail and delay or lose messages. Again, a good solution should be able to provide continued service – possibly of a degraded quality – despite the occurrence of failures of the infrastructure.

With the above arguments in mind one can argue that a key tool to address the problem of EAI is a specification language and a programmatically way that address the two issues mentioned above.

The Model Driven Architecture (MDA) is a promising software engineering approach suggested by the Object Management Group (OMG) [14]. Its aim is the automation of the software cycle which normally includes design, implementation, deployment, integration, re-design, re-implementation, etc. It relies on the use of tools (as opposite to the conventional manual approach) along the different stages of the software cycle. Central to the MDA are the concepts of model, metamodel and abstraction levels. In brief, a model is the specification of the system under construction, at a given level of abstraction; whereas the metamodel can be a Domain Specific Language (DSL) used to specify the system. The main abstraction levels, in a descent way, are: computation independent (CIM), platform-independent (PIM), platform-specific (PSM) and deployment level. All them provide different models to describe the solution.

A good alternative to address the first challenge presented above is a DSL with built-in constructs to capture the most fundamental concepts involved in EAI solutions such as messages, communication pipes and processing filters, but at a high-level of abstraction. For instance, such languages can be used to describe solutions by means of PIMs, that is, specifications that are implementation neutral and can be programmatically transformed into executable code. Equally important, to address the second issue, such a language should offer a means for capturing exceptional situations likely to have an impact on the EAI solution. To the best of our knowledge and in accordance with

results from previous research [6], DSLs with these highly desirable features are still a research topic. We are aware that there are some preliminary results in this direction. For example, in [6] the authors present Guaraná – a DSL designed to produce PIMs; in other words, it addresses the first issue hinted above; a limitation of Guaraná is that as it is, it can only specify normal execution flows; in other words, it does not address the second issue since it does not have any mechanisms to specify exceptional situations. As an alternative to cover the gap, we suggest the enhancement of Guaraná with constructs for detection and handling of exceptional situations. In pursuit of this goal, we show in this paper how Guaraná ports can be enhanced to signal exceptions when communication operations (e.g. read, write, solicit) executed by ports against an asset fail to complete, or a validation test on received data produces abnormal results. We target port operations first because we consider that they are the most fault-prone operations in an EAI solution. Also, we show how a conventional Event Condition Action (ECA) mechanisms can be used to handle these exceptions. Several engineering requirements combine to make EAI a hard problem; before tackling them, it is worth clarifying what requirements we take on board in our research and what assumptions we make.

The first requirement we account for is that the existing assets are and should continue to be functionally independent from each other with and without the EAI solution in operation. Mutually-dependent assets fall outside the scope of this paper. It follows that our EAI solution should provide only exogenous coordination. To meet this requirement we rely on loosely-coupled interactions between the solution and the assets.

Another typical requirement on the EAI solution is that it should not involve the modification of the code or configuration of the original asset; the implication of this restriction is that the EAI solution can count only on the original interfaces that the participating assets offer, to make them interoperate.

We make no assumptions about the physical locations of the assets. They can be located within the same building and be linked by a LAN or in different continents and communicate over the Internet. The involvement of the Internet presents the designer with additional challenges: Internet communication is far from being fully reliable; it can lose and duplicate messages; and introduce unpredictable delays. Without due attention, these impairments can render an EAI solution unoperational. Communication delays become highly relevant in EAI solutions with strict and tight time constraints.

Since we focus on EAI, we can leave authentication and security issues out of the equation as these problems are not of major concern in these scenarios. Likewise, we can assume that access to data between assets is always granted, likely under some restrictions of no relevance to our discussion.

This paper is structured as follows: Section 2 places our research in context and discusses the related work; Section 3, offers a brief introduction to our DSL Guaraná and places it within the context of the model driven approach; Section 4, is the heart of our paper – it discusses our failure semantics and shows how Guaraná's ports can handle exceptional situations; Section 5, presents a realistic scenario of enterprise integration that we use as a validating example, and discusses our event condition action-based exception handling mechanism to support Guaraná. Finally, we draw conclusions and future work in Section 6.

2 Related Work

The UML-profile [13] for EAI solutions is directly related to our work on Guaraná and is arguably an attractive alternative. We rule it out on the basis that UML profiles are basically extensions to UML intended to cover the limitations of the native UML; unfortunately, the UML-profile for EAI solutions has not been very successful due to its complexity and lack of expressiveness (see [1] for a discussion on the adequateness of using UML-profiles to represent DSLs).

The Business Process Modeling Notation (BPMN) [18] can be used for specifying EAI solutions but at CIM level, that is, at a level that is too abstract for EAI designers. Related to our work are also EAI technologies like BizTalk [10], Mule [12] and Camel [5] which produce technology-dependent solutions: in the context of MDA, BizTalk fits the platform-specific level while Mule and Camel belong to deployment level. We regard BPMN, BizTalk, Mule and Camel as complimentary to Guaraná, rather than competitive.

Our work is closely related to the on-going research on exception handling in Web services composition. Of interest to us is the discussion presented in [19]. We share with the authors the idea of using ECA policies to handle exceptional events. However, the paper studies the problem at BPEL level of abstraction and suggests the use of an integrated exception handling mechanism with the intention of conducting execution planning to prevent the occurrence of exceptions. Our goal is different – we address exceptions at PIM level of abstraction and propose mechanisms to recover from exceptional situations rather than to prevent their occurrences. A similar discussion on exception handling at BPEL level and complimentary to [19] can be found in [9]. In [4] the authors propose a policy-driven middleware solution (implemented in .NET and manually portable to other platforms) to handle exceptions in web service composition; we consider this a valid result but too implementation specific, since our interest is in abstract PIMs. With this paper we share the view that communication operations are the most fault-prone. Relevant to us is also the classification of faults which can roughly be mapped into the exception that Guaraná's ports can detect.

An illuminating discussion about the complexity of handling exceptional situations in EAI, such as the unexpected cancellation of an operation due to infrastructure failures or human related events, is presented in [7]. Authors argue that to be effective, a compensation mechanism should take into consideration the state of the two interacting applications. As the discussion is at conceptual level, the authors present no solution.

3 An Overview of Guaraná – a Domain Specific Language for EAI

A DSL is a well focused language developed to address problems in a particular domain. It provides a set of dedicated abstractions, elements and notations with formalization to assist the designer in expressing its solution at the level of abstraction of its DSL. In MDA, a given model is programmatically transformed into a model of a lower level of abstraction. Models of high-level of abstraction are implementation neutral and called PIMs; whereas models of low-level of abstraction are implementation specific and called PSM. There are estimations [17] that show that for each dollar spent

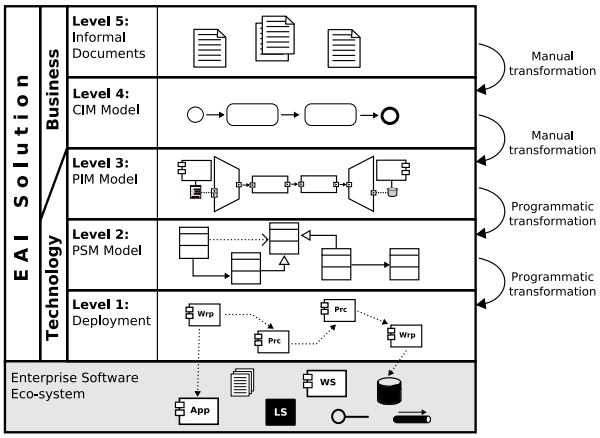

Fig. 1. Abstraction levels for an Integration Solution.

on developing an application, companies usually spend from 5 to 20 dollars to integrate it into EAI solutions. From this perspective, the MDA approach looks like a promising alternative to cut these costs down. For instance, the designer can produce a PIM model of the EAI solution that can be re-used to automatically derive as many PSMs as necessary to match technology evolution within the enterprise.

The feasibility of the MDA approach depends on the availability of the source metamodel, target metamodel, transformations, etc. For instance, in the EAI domain a DSL is still a miss. We suggest Guaraná as an alternative to cover the gap. Guaraná produces graphical designs of EAI solutions at a high-level of abstraction; it uses Enterprise Integration Patterns (EIP) [8] and covers currently existing gaps in the field.

3.1 Abstraction Levels for Integration Solutions

In the MDA, the specification of an EAI solution can undergo several transformations through different levels of abstractions before it is finally implemented on top of the software ecosystem. This idea is shown in Fig. 1 with the intention of placing Guaraná within this context. The figure shows five levels of abstraction separated into business and technology. Level 5 is the most abstract one and is entirely technology-specific solution neutral in contrast, level 1 is the final technology-specific deployable solution. Guaraná belongs to the third level.

LEVEL 5: Models are produced by business analysts and provide an informal description of the problem at a high-level of abstraction, that is, with no notion of applications, message flows or technology like software ecosystems. Models are specified in natural language and normally suffer from imprecisions, omissions and ambiguities.

LEVEL 4: Here models are considered CIMs. They are refinements of models from level 5, produced manually by business analysts and expressed in standards like BPMN [18]. The notions of participating applications (source and consumer of data) and message flow appear at this level; yet the model does not capture core domain

specific design concepts like the internal structure of the solution or the communication with its applications. The absence of these concepts prevents their programmatic transformation into executable models.

LEVEL 3: Models of this level are produced from models of level 4 manually by system analysts with expertise in EAI solutions. They precisely specify the functionality and structure of the solution in a manner that it can be programmatically converted into PSMs of level 2. So they deal with concepts at the granularity of applications, processes, tasks, message flow, integration links, ports, wrappers, etc. These models are expressed in a DSL with built–in constructs to describe all the EAI specific concepts listed above.

LEVEL 2: Models of this level result from automatic conversion of models from level 3; they are PSM and can be mapped into executable code of the chosen technology.

LEVEL 1: Models of this levels are the actual executable code of the solution and are programmatically generated from models from level 2.

3.2 Guaraná Constructors

Guaraná provides a set of domain specific constructors to design EAI solutions. This language not just introduces and describes this domain specific concepts, but also provides a very expressive and needful graphical notation for this constructors, that allow to visually design an EAI solution. Below we introduce the main constructors and in Fig. 3 we provide an example of a designed EAI solution with Guaraná where the use of these constructors can be seen. **Decorators** are used to provide visual information about the participating assets and their layer(s). They do not have an influence on the executable model. **Building blocks** represent the processing constructors of the EAI solution and are composed of tasks. There are two types of building blocks in Guaraná: wrappers and processes. A **wrapper** communicates the EAI solution to an asset, so it contains communication-specific tasks and has a port connected to a decorator. **Processes** model the essential services of the EAI solution, so they contain integration-specific tasks; they are connected (through ports and integration links) to each other and/or to wrappers. **Slots** are memory buffers used within building blocks for port to task and task to task internal communications. **Tasks** are message processing constructors and appear inside processes and wrappers. A task reads messages from incoming slots, processes them (e.g. enriches, translates, filters, etc.) and deposits the result in the outcome slot. **Ports** are used to communicate the internal building blocks of an EAI solution and the EAI solution with its assets. **Integration links** are channels that transport messages between building blocks. They are used to connect the entry/exit ports used by building blocks.

4 Failures in Application Integration Solutions

The general assumption we make about the reliability of the components involved in an enterprise application solution is that they will occasionally fail. Thus our goal is to provide the enterprise application integration with mechanisms to tolerate the occurrence of failures as opposite to prevent their occurrence. For this to be possible, we need to identify the failure behavior that the enterprise application integration are likely

to exhibit. The failure behavior is also known as failure model or failure semantics [2] and stipulates what kind of errors the system (the enterprise application integration for instance) will be able to tolerate: detect at runtime, execute corresponding recovery action and return to the normal execution flow possibly with a degraded performance. An application EAI solution can fail in different ways, non-surprisingly, different authors suggest different classes of failures (see for example [4], [9]); yet it seems to be a general consensus that most of the failures that impact EAI solutions emerge from the execution of operations that involve exchange of data with assets; the reason being that these operations normally involve network communication and possibly over unpredictable channels like the Internet. The EAI solution can run distributed in different machines, so in this case network problems also should be considered for building blocks' communication. This type of errors are not addressed in this paper. On this basis and to comply with space restrictions, this paper focuses only on two types of failures that might occur inside ports used by the EAI solution to communicate with its assets: omission failures and response failures. Yet it is worth mentioning that our idea is general enough to be expanded to other operations executed by the EAI solution.

Omission Failures (OMF): In our communication model we assume that once a communication operation (read, write and solicit-response) is started by the EAI solution, it terminates within a strictly defined time interval and declared by the EAI solution either success or failure. The failure result models situations in which the network and asset problems might prevent the solution to send or receive a piece of data to/from the asset within the deadline interval, when this happens we say that the asset has exhibited an omission failure. Notice that in our communication model operations completed beyond the time constraint are taken as failure, so data received by a read operation after the expiry of the deadline is ignored. In our discussion we use *OK* and *OMF:NOK* to represent success and failure, respectively.

Response Failures (REF): As suggested by leading standards in e-business middleware like RosettaNet [15] and/or by well known integration technologies like BizTalk [11], it is not enough to receive a response in time as the responder, the asset in this case, might respond incorrectly. Thus a received message has to satisfy some syntactic validation tests (e.g., headers and body inspected and understood) before it can be taken by the EAI solution for processing. This kind of failures are known as response failures. To model them we run a validation test on every message received by the EAI solution, that produces either success or failure. Again, we use *OK* and *REF:NOK* to represent success and failure, respectively.

4.1 Exceptions of Guaraná Ports Operations

Guaraná provides four types of ports for communication: one-way EntryPort, one-way ExitPort, two-way SolicitorPort and two-way ResponderPort. One-way EntryPorts are used for reading messages in an internal EAI solution communication and from assets, as well; one-way ExitPorts are used similarly, but for writing. Two-way SolicitorPorts are used to solicit data from assets in solicit-response mode; in principle this operation can be split into two individual operations or abstracted as single atomic one; for simplicity we discuss only the latter case. Two-way ResponderPorts are irrelevant in our arguments and not discussed further.

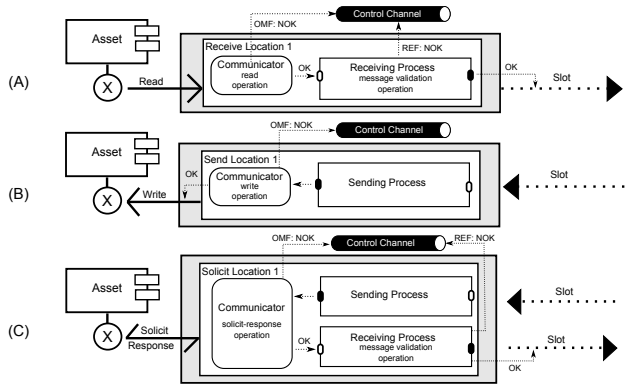

Fig. 2. (A) Entry Port, (B) Exit Port, (C) Solicitor Port.

The instrumentation of our failure model in Guaraná's ports is shown graphically in Fig.2. EntryPort and SolicitorPorts can contain one or more locations with their respective communicators. Each location is associated to a single source of data (e.g., application layer such as database, file, user interface). We assume the existence of a single location as this is enough to explain our ideas. The actual communication operation (read, write and solicit-response) is performed by the communicator; so to implement our omission failure model, we provide communicators with the notion of deadline to complete their operations. Read messages are delivered by communicators to the receiving processes. The sending processes are irrelevant in our discussion yet we can mention that they process the message destined to assets, before passing it on to the communicators. To model response failures, we include a validation operation inside receiving processes. As show in the figure, in response to a given operation (e.g., take read operation of EntryPort A) communicators produce either OK or OMF:NOK. The OK message represents the normal response and is fed into the normal flow, whereas OMF:NOK represent the abnormal result and is notified to the control channel. Similarly, the validation operations produce either OK (fed into the normal flow) or REF:NOK which is notified into the control channel.

5 Validation

To validate our ideas we will show how both the normal and exceptional execution flow can be specified in Guaraná.

5.1 Example

To set the scene, we will use the scenario of an application integration problem under study at Unijuí, Brazil. Apart from some small modification introduced to highlight the issues of our research interest, the scenario is realistic. The project involves five assets: a Call Center System (CCS), Payroll System (PS), Human Resources System

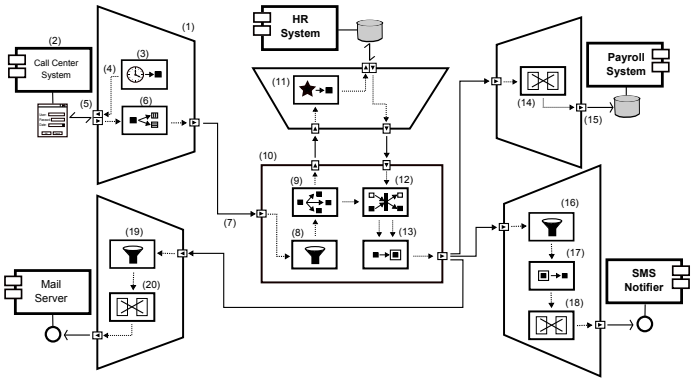

Fig. 3. UNIJUÍ's integration solution with fault-tolerance.

(HRS), E-mail Server (ES) and Short Message Service (SMS). The CCS manages the university telephone system. The PS processes the monthly payments of the employees. The HRS manages the personal data of the employees. The ES runs the e-mail service. SMS runs a service for sending text messages to mobiles phones. The problem can be informally described as follows:

1. Employees use the phone facilities of the university for making external phone calls of both business and personal purposes. All calls are recorded by CCS.
2. Business calls are free. The cost of personal calls is deducted from the caller's salary which is processed by the PS on the last day of the month at 9:00 a.m.
   – *No calls can be deducted before notifying the caller by e-mail, SMS text or both.*
3. To deduce the cost of a call, the PS requires: 1) the caller's name, personnel number, e-mail and mobile phone number, from the HRS; and 2) the cost and destination of the call, from the CCS.
4. To guarantee that a call will be deducted from the current month's salary the PS needs to receive the input by 8:00 a.m. on payment day.
   – *Input received after this time is logged and processed in the next month.*

The description has two salient features: 1) It includes operations with strict time constraints, for example, "input by 8:00 a.m. on payment day". 2) It accounts for potential exceptional situations and separates the normal execution flow (normal text) from the exceptional one (italic text). This problem can be solved by using the exception mechanism introduced in Section 4.

5.2 Integration Solution

The Guaraná specification of our example is shown in Fig. 3; ignore for the time being, deadline constraints and potential exceptions. The integration flow is started by the timer task (3) located inside the wrapper (1) that communicates with the CCS – represented by a decorator (2). This task creates an activation message every t units of time (e.g. five minutes) and writes it to a slot (4). The message activates a solicitor

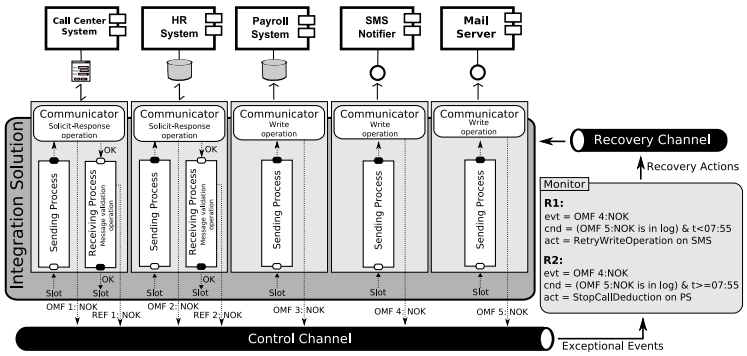

Fig. 4. ECA - based exception handling mechanisms.

port (5) that extracts all phone calls made in the last five minutes. The CCS offers only a user interface, so the solicitor port uses a scrapper (a technique for extracting information through a user interface) to interact with the CCS. The splitter task (6) breaks the message into individual messages containing just one phone call each. These messages are sent, through an integration link (7), to the central process (10). Messages related to irrelevant (e.g., free or insignificant cost) calls are filtered out by a filter (8). Messages of interest are replicated (9) to the wrapper of the HRS and to a correlator (12). The HRS is queried by the message created by a custom task (11); the response from the HRS is sent to the central process (10) where it enriches the original message by means of an enricher (13). The enriched message is sent after the corresponding translations (14, 18 and 20) to match destination formats, to the PS, SMS and ES. Messages that, for some reasons, do not contain a phone number or an e-mail address are filtered out (respectively by filters 16 and 19). A slimmer (7) shortens messages down to the short text standard before sending them to the SMS.

5.3 An ECA-based Exception Handling Mechanism

To handle exception occurrences at programming level and independently from the normal execution of the EAI solution, we suggest the use of an exception handling mechanism based on the conventional Event Condition Action (ECA) paradigm. We show its functionality with the help of Fig. 4 which is a simplified version of Fig. 3 meant to highlight port operations between the EAI solution and the five assets involved.

Our mechanism involves a monitor and a control channel that links the monitor with the EAI solution (say by a publish/subscribe paradigm). Also, the monitor is linked to the EAI solution by the recovery channel for recovery actions. The EAI solution represents the normal execution flow (normal text of our validating example) whereas the monitor represents potential exceptional executions (italic text of the example). The monitor is instrumented with ECA rules that execute exceptional actions upon receiving exceptional events through the control channel. As shown in Fig. 4, exceptional results (OMF:NOK and REF:NOK) from the execution of port operations (solicit-response, message validation and write) are published to the control channel. The normal results

of the operations are fed only into the normal execution flow, but if necessary, they can also be published to the control channel to be used by the monitor.

The exceptional actions are application-specific and fall outside the scope of this paper; yet we can briefly mention that they are recovery actions whose execution brings the control flow back to the normal execution. For instance, an operation can be simply ignored, retried, etc. Some typical recovery patterns are discussed in [9] and [3]. The rules can be written in a rule language (see for example [4]) and executed by a conventional rule engine supported by timers, event-log files, queues, etc., (see for example [16]). The discussion of these details falls outside the scope of this paper. The rules shown inside the monitor are only illustrative and far from being complete; they are meant to show how exceptional situations can be handled; to save space we will focus only on the exceptional execution flow of point 2 of our example: *No calls can be deducted before notifying the caller by e-mail, SMS text or both*.

In our simplified notation R, evt, cnd, act and t stand for rule, event, condition, action and time respectively. The $07:55$ represents the time on the payment day to notify the PS to stop it from processing a call deduction when the caller has not been notified. Similarly *OMF 5:NOK is in log* checks for the existence of OMF 5:NOK records in the log file of the rule engine. $R1$ captures the possibility that the *write* operation against the SMS fails; the condition checks if the notification by e-mail has failed and if there is time to retry the SMS operation; when the exceptional event OMF 4:NOK is received and the condition holds, the recovery action *RetryWriteOperation on SMS* is executed. $R2$ is complimentary to $R1$ as it also reacts to the OMF 4:NOK exceptional event; yet it is triggered when it is time ($t >= 07:55$) to notify the PS of the problem and executes the *StopCallDeduction on PS* recovery action to stop the PS from processing the call under question.

6 Conclusion and future work

It is worth emphasizing that at the current stage of our research the failure semantics of Guaraná's ports consider only omission and response failures that may be raised due to a local time constraint at port level when ports interact with assets. Our future work is to enhance Guaraná's failure semantics also with the capability of capturing message processing failures that may occur into the sending process element at ports (see Fig. 2). This element can be used to validate the message before forwarding it to the communicator so that assets are prevented from receiving and processing invalid messages that will only produce NOK outcomes and exception signals. Another idea is to extend this idea to the whole EAI solution and thus considering those message processing failures that may occur within building blocks, since the principle here is the same as for sending processes; in principle each building block involved in the flow might produce success (OK) or failure (NOK) after processing a message.

Considering the whole EAI solution, the failure semantic could be enriched with the notion of a global deadline (global time constraint). This kind of constraint should specify a time-to-live for messages, meaning the message is valid and can be normally processed by the EAI solution, as well as, be delivered to the target asset(s) within this time. In this case a corresponding class of failure could be raised and handled, if the EAI

solution did not meet the global time constraint. There would be a direct relationship between these two kinds of constraints, since the total amount of time in local time constraint could give a hint to build the global time constraint.

The paper recognizes the need to account for exceptional situations that normally impact EAI solutions at run time and suggests an approach to capture them at an abstract level of the specification. To address the problem and in support of the model driven approach to cope with computer technology evolution, the paper contributes with an innovative DSL that 1) produces PIMs; and 2) whose operation (ports at current stage) invocations account for exceptional outcomes: they either produce a normal result or an exceptional event that is processed by an event condition action-based monitor that triggers recovery procedures.

References

1. A. Abouzahra, J. Bézivin, M.D. Del Fabro, and F. Jouault. A practical approach to bridging domain specific languages with UML profiles. In *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA*, volume 5, 2005.
2. F. Cristian. Understanding fault–tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.
3. V. Ermagan, I. Kruger, and M. Menarini. A fault tolerance approach for enterprise applications. *Services Computing, 2008. SCC '08. IEEE Int'l Conf on*, 2:63–72, July 2008.
4. A. Erradi, P. Maheshwari, and V. Tosic. Recovery policies for enhancing web services reliability. In *Proc. Int'l Conf. Web Serv.*, pages 189–196, 2006.
5. Apache Foundation. Camel Home, 2008.
6. R.Z. Frantz, R. Corchuelo, and J. González. Advances in a DSL for Application Integration. In *Proceedings of the Zoco'08 Workshop*, pages 54–66, Gijón (España), 2008.
7. P. Greenfield, A. Fekete, J. Jang, and D. Kuo. Compensation is not enough. In *EDOC '03: Proceedings of the 7th International Conference on Enterprise Distributed Object Computing*, page 232, Washington, DC, USA, 2003. IEEE Computer Society.
8. G. Hohpe and B. Woolf. *Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003.
9. A. Liu, Q. Li, L. Huang, and M. Xiao. A declarative approach to enhancing the reliability of bpel processes. In *Proc. IEEE Int'l Conf. Web Services*, pages 272–279, 2007.
10. Microsoft. Microsoft BizTalk Server 2006 R2 Home, 2008.
11. Microsoft. BizTalk Framework 2.0: Document and Message Specification, Dec 2000.
12. Inc. MuleSource. Mule 2.x User Guide, 2008.
13. Object Management Group (OMG). OMG EAI Profile Home, 2004.
14. Object Management Group (OMG). OMG Home, 2009.
15. Rosettanet. Rosettanet: Implementation framework–core specification, version: V02.00.01, revised 6 mar, 2002.
16. M. Strano, C. Molina-Jimenez, and S. Shrivastava. A rule-based notation to specify executable electronic contracts, cs–tr no. 1115. Technical report, School of Computing Science, Newcastle University, 2008.
17. J. Weiss. Aligning relationships: Optimizing the value of strategic outsourcing. Technical report, IBM, 2005.
18. Stephen A. White. Business Process Modeling Notation (BPMN) Specification 1.0, 2009.
19. L. Zeng, H. Lei, J j. Jeng, J-Y. Chung, and B. Benatallah. Policy–driven exception-management for composite web services. In *Proc. Seventh IEEE International Conference on E–Commerce Technology (CEC05), 19–22 July*, pages 355–363. IEEE Computer Society, 2005.