The effects of exceptions on enterprise architecture

Bas van Gils[*] basvg@cs.ru.nl

Dovilé Vojevodina[†] d.vojevodina@erp.lt

February 9, 2006

Abstract

Exception handling in enterprises is an increasingly important topic since exceptions may seriously disrupt day to day operations which may impact on the profits of the enterprise. In this paper we explore two views on the enterprise (top down: architecture, and bottom up: workflow) with respect to exception handling. More specifically, we focus on using both views on the enterprise to handle exceptions in a structured and efficient manner.

1 Introduction

Good and stable relationships with customers is getting increasingly important in today's networked world of E-commerce and service orientation. According to some, achieving a high level of customer loyalty is the key to successful online business (See e.g. [Pastore, 2000]) which can only be achieved by focusing on customer satisfaction, once reputation with regard to performance, quality of information etcetera (See e.g. [Turban et al., 1999]). This is particularly true for *business to business* interactions. Others claim that the only thing that matters is price, since searching on the Web for "better deals" is relatively cheap and one should thus focus on keeping the price of products and services as low as possible since (potential) customers / partners can easily switch. This is probably mostly true for *business to consumer* interactions.

All in all, enterprises are in a tight spot in a world where change is a constant an the competition is high. Many different approaches are used and proposed to deal with the complexity of today's business such as *Architecture* and *workflow*. Following the standard definitions from literature these can be defined as follows:

Definition 1.1 (Architecture) *The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principle guiding its design and evolution.*

—[IEEE, 2000]

Definition 1.2 (Workflow) *The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.*

—[WfMC, 1999]

Architecture, thus, looks at enterprises at a high level of abstraction, whereas workflow provides a more detailed view (of the processes in enterprises). As such they can can be seen as complimentary.

[*]Radboud University Nijmegen, Institute for Computing and Information Sciences, Nijmegen, The Netherlands, EU

[†]Inoformation Technology Department, Vilnius Gemiminas Technical University, Vilnius, Lithuania, EU

There are several reasons / drivers for making an architectural description of enterprises (i.e. a document describing the architecture of the enterprise). One of the most important 'internal drivers' is the alignment of business and IT. Another internal drivers is to use the enterprise architecture as a management instrument. Obviously there are also some 'external drivers'. These are mainly focused on themes such as B2B integration of applications, offering customers direct access to processes using E-services and also for regulation purposes (governmental institutions often want insight in (large) enterprises). It seems apparent that architecture is, thus, one way of dealing with complexity of the changing enterprise and its environment since it provides a stable view of its essential properties. In other words, architecture can be seen as a top-down view on the enterprise.

On the other hand, workflow provides the "micro-management view" of the processes in an organization; it can be seen as a bottom-up view on the processes of the enterprise as it specifies not only the tasks that have to be performed in a certain process, but also the order in which they are performed, the assignment of tasks to resources, the documents involved etcetera. If workflow is designed in a flexible and structured way across the entire enterprise (and possibly even its partners) then this, too, helps in dealing with the uncertainty and changing nature of the environment. Especially when one acknowledges that *exceptions* in workflow may occur and that these can have a big impact on the outcome of workflow. Different definitions of these exceptions are used in literature such as e.g. [Zongwei et al., 2000, Hwang et al., 1999]. In this paper we will use the following definition:

Definition 1.3 (Workflow exception) *Any business activity that cannot be executed in a predefined manner is referred to as an exception.*

—[Vojevodina and Kulvietis, 2005]

These exceptions can be classified in numerous ways as we will discuss later in this paper. It is interesting to observe that, when designing the workflow for certain processes then possible exceptions are often incorporated in the design already (in this case we no longer speak of an exception, but a *business case*. A deviation from normal workflow is only an exception if it comes unexpected). In this position paper we will study the relation between workflow exceptions and enterprise architecture. More specifically we will try to answer the following questions:

- How can architecture help trace and deal with workflow exceptions?
- How do workflow exceptions influence the enterprise architecture?

Our ambition for this position paper is not so much to come up with well-tested solutions that can readily be applied to business. Instead, we wish to study the issues at hand and present our initial ideas around the issues presented above.

The remainder of this paper is organized as follows. In Section 2 we will present a brief overview of some of the aspects of the available literature on architecture. The main focus of this section is to provide a high-level insight in what architecture is, what architectural models are and what the role is of principles, guidelines, standards etcetera. We will also present the running example for this paper in this section. In Section 3 we will present a more detailed view on workflow and present a classification for workflow exceptions which we will relate to the aspects of the architecture definition. In Section 4 we will make our main contribution analyze what happens when workflow exceptions occur and what their impact is. This section will provide the material to finally answer our research questions in Section 5

2 Architecture

The aim of this Section is to give an overview of literature on enterprise architecture. The material presented in the section is mainly based on / inspired by [Lankhorst, 2005, Rijsenbrij, 2004, Franken and Janssen, 1998, Zachman, 1987, TOGAF, 2004]. In the following the reader should be aware that we make a distinction between the actual architecture (as perceived by a person) and an architecture description (what s/he puts on paper). From the context it is usually clear which one we mean.

The IEEE-definition for architecture is given in Definition 1.1. This definition has two important aspects, that of *principles* and of *blueprint*. These aspects should therefore also be present in a definition of *enterprise architecture*. It is interesting to observe that different approaches to (enterprise) architecture tend to focus on one of these aspects. For example, Rijsenbrij has the following definition of architecture:

> Architecture is a coherent and consistent collection of principles, specialized into rules, guidelines and standards which describe how the enterprise its information infrastructure, applications and technical infrastructure are designed and used in practice.

This definition obviously focuses on the *principle* aspect. On the other hand, in the ArchiMate approach. In other approaches such as for example Testbed, the focus is more on architectural models (i.e., the *blueprint* aspects). The definition of enterprise architecture that we will use is:

Definition 2.1 (Enterprise Architecture) *A coherent whole of principles, methods and models that are used in the design and realization of enterprise organizational structure, business processes, information systems and infrastructure.*

— [Lankhorst, 2005]

In essence this definition states that enterprise architecture clarifies what an enterprise looks like (the models) and why (the principles). The most important characteristic of enterprise architecture is that it provides the insights needed to balance the requirements that the highly agile and rapidly changing business poses on the (technical and informational) infrastructure of the enterprise.

Given the research questions of this paper we will focus mainly on the models, event hough we do acknowledge that the principles guiding the design of these models are important. As an example of this importance consider the principle "Customers can do business with us anytime, any place, anywhere". This clearly poses some restrictions on the design of the enterprise, since the customer must be able to interface with the enterprise using different technologies (such as phone, a web-browser or fax). Even more so, it also poses restrictions on the technical infrastructure since 24/7 availability is required!

One of the goals in the ArchiMate approach is the integration of architectural domains (such as information architecture, process architecture, technical architecture, application architecture and product architecture) to provide a holistic view of the enterprise.

Architecture models (and modeling in general) are all about abstraction. It is of great importance that architectural models use a similar level of abstraction for them to make any sense. To this end, the service concept is introduced:

Definition 2.2 (Service) *A unit of functionality that some entity (e.g., a system, organization, or department) makes available to its environment, and which has some value for certain entities in the environment (typically the 'service users').*

The service concept makes it possible to abstract from physical entities (such as hardware, or a specific customer), and leads to a layered view of enterprise architectural models. From the definition of the service concept it follows that services are offered and used. As such they form the 'glue' between the layers of architectural models. These layers are[1]:

[1]The three layers presented here come from the ArchiMate project. Other approaches may use a different set. For example, four layers (business, information, application, infrastructure) are used in [Rijsenbrij, 2004].

business layer : offers products and services to external customers, which are realized in the organization by business processes.

application components : supports the business layer with application services which are realized by some (software) application components.

technical infrastructure : offers infrastructural services needed to run applications, realized by computer and communication devices and system software.

This is illustrated in Figure 1 where the dotted arrows denotes the *realization* relation and the solid arrows denote the *used by* relation relation. Different concepts are used in each layer, each

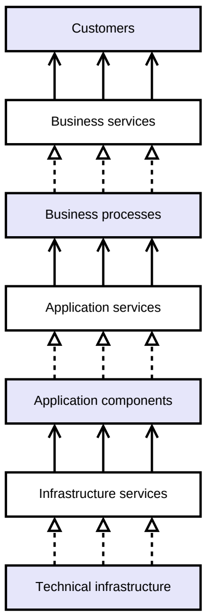

Figure 1: Layered view of architectural models

requiring their own notation. The notation used in ArchiMate is quite loose and intuitive. In the discussion here we will discuss the individual concepts and illustrate their notation by means of the example architectural model using the ArchiMate language presented in Figure 2. This example is taken from the ArchiMate book [Lankhorst, 2005]. We chose not to modify it since this model is already concise and serves our purposes well.

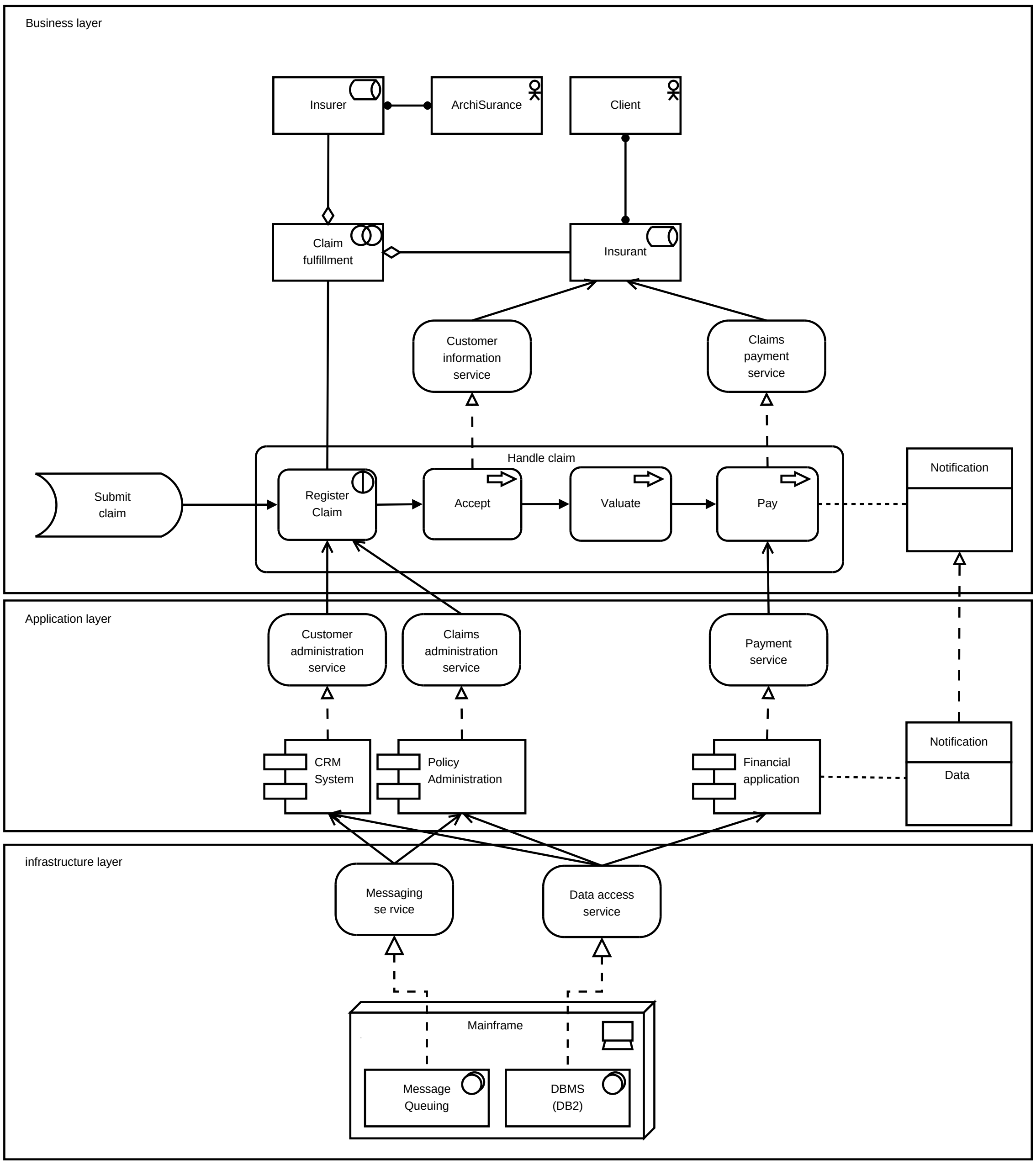

Figure 2: The "handle claims" example

In the business layer we use the following concepts[2]: *Business actor* (*ArchiSurrance* and *Client*), *Business role* (*Insurer* and *Insurant*), *Business collaboration* (*claim fulfillment*), *Business interface*, *Business object* (*Notification*), *Business service* (*customer information service* and *customer payment service*), *Business activity* (*Accept*, *Valuate* and *Pay*), *Business process* (*Handle claim*), *Business function* (*Customer relations* and *Financial Handling*), *Business interaction* (*Register claim*) and *Business event* (*Submit claim*). Furthermore, the ArchiMate language allows the creation of more high level business concepts such as *product*, *contract*, *value* and *meaning*. Neither of these are illustrated in the example.

[2]In the following text we will include references to concepts from the example in Figure 2 in a slanted font between parenthesis. If no references are included for a certain concept then it is not illustrated in the example.

Observe that we will get back to the discussion of business activities in Section 3 where we present our views on workflow. Even more, some of the definitions used in the ArchiMate language seem a bit awkward. For example, a business object is defined as:

> The passive entities such as a business process or functions that are manipulated by behavior.

More elaborate and more widely used definitions of business objects exist. See for example [Gils, 2002] or [Heuvel, 2002] for an overview.

In the application layer we use the following concepts: *application component* (*CRM System*, *Policy creation* and *Financial application*), *Application collaboration*, *Application interface*, *Data object* (*Notification*), *Application service* (*Customer administration service*, *Claims administration service* and *Payment service*), *Application function* and *Application interaction*.

It is interesting to observe that most of the business events are supported by application components (but not all), and that some business events need more than one application components. Furthermore, this clearly illustrates once again that services form the glue between the layers in the model.

In the technology layer we use the following concepts: *Node*, *Infrastructure interface*, *device* (*Mainframe*), *system software* (*Message queuing* and *DBMS*), *communication path network*, *infrastructure service* (*Messaging service* and *Data access service*).

The ArchiMate language is, obviously, not the only language/ notation that exists for creating architectural models. We concur with [Lankhorst, 2005] that, apart from the ArchiMate language, no languages exist primarily for representing enterprise architectures. Most languages in existence today (such as UML [UML, 2003], IDEF [Mayer et al., 1995] and Testbed [Franken and Janssen, 1998]) are primary focused on software systems. This does not mean, however, that these languages can not be useful for enterprise modeling.

The main advantage of an *integral* enterprise architecture approach over more domain specific approaches (for software or business systems) is the fact that it enables the analysis of (the impact of) change in the enterprise:

quantitative analysis : by performing quantitative analysis insight can be gained in *performance*, *cost* and *reliability* of systems.

functional analysis : provides more insight in how the architectural description is to be interpreted and what is meant by particular entities or relationships, thus providing a way to validate the correctness of the architectural description.

We believe that architectural descriptions can, and should, also be used for dealing with exceptions. In Section 4 we will get back to this issue.

3 Workflow & exceptions

The main purpose of this section is to introduce the concept of workflow and to explain the relation with enterprise architecture with a strong focus on exceptions. We will first present our view on workflow and then shift the focus to exceptions and exception handling.

3.1 Workflow in Enterprise Architecture

The deployment of workflow as a method for managing real world situations is highly dependent on the context (i.e. the *environment*) of the organization. Workflow as a technology has been used in various IT products for a number of years, especially in group-ware systems. Since then, workflow*systems* have evolved in a separate and important IT product by itself. The very nature of workflow from the beginning was to manage electronic documents / information and enterprise resources by particular business rules and in that way make people do what they must do in a particular time, with particular information and in particular manner.

The definition for workflow that we use in this paper was presented in Definition 1.2. Simply put, it is the automation of a business process in whole or in part which, if done right, is beneficial to the enterprise. The main purpose of workflow is to improve efficiency of the execution of processes. There are, however, many other benefits of using workflow technology such as reducing the time cycle required to perform the task, productivity, improved customer service through cycle time reduction and process control, better ways for collaboration and knowledge sharing [Lotus, 1999, Hollingworth, 1995].

Every enterprise (or organization, for that matter) has a different architecture. The workflow applications supporting the processes in organizations mainly focus on production, enterprise, collaboration or customers. Furthermore, they may involve both internal and external enterprise processes which define tasks arrangement, business rules and resources. Some authors also distinct one more workflow application type, ad-hoc, which is simply a combination of all, as mentioned earlier, but to the best of our knowledge there are no workflow management systems that actually support this type of workflow processes.

In the enterprise architecture model, workflow is the most related to business layer. However, it is important to realize that workflow has an impact on all enterprise layers, if only because workflow management systems live in the application layer. These applications, in turn, run on on systems which live in the infrastructural layer. Our definition of a workflow system is:

Definition 3.1 (Workflow Management System) *A system that completely defines, manages and executes "workflows" through the execution of software whose order of execution is driven by a computer representation of workflow logics.*

— [Hollingworth, 1995]

In other words, workflow management system acts as the glue (similar to the services between the enterprise architectural layers) between services which compose the enterprise architecture itself.

The main ways for constructing/depict workflow are flow diagrams [Sharp, 2000] and Petri-nets [Aalst, 2000]. As enterprise architecture is a whole which depicts the organizational structure, business processes, information systems and architecture, we will use flow diagrams for depicting real world situations and change in enterprise architecture, as Petri-nets are more suitable for constructing working workflow models in the context of workflow management system. Consider, for example, the payment process for an online shop as illustrated in Figure 3. A simple

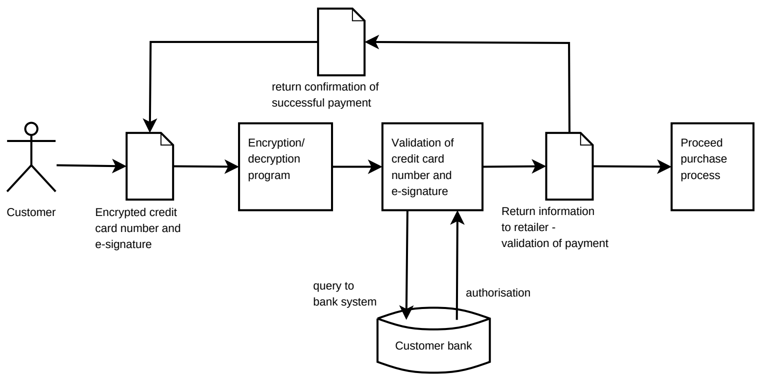
return confirmation of successful payment
Customer
Encrypted credit card number and e-signature
Encryption/ decryption program
Validation of credit card number and e-signature
Return information to retailer - validation of payment
Proceed purchase process
query to bank system
authorisation
Customer bank

Figure 3: example: payment process

process of customer making payment on e-shop can be described as workflow. This small workflow is usually modeled as a sub-workflow and included into the main goods purchase business process workflow. The very nature of workflow is to do all tasks in sequence and return needed information to previous tasks or actors (if they are the previous point) needed to make other business process steps or decisions. in this example customer is sending encrypted credit card credentials to the retailer e-business system, which further makes verification and validation of these credentials using third part system (in this case bank authorization system). Every step is executed passing needed data or query to the next task or program (which performs a task). The result of credit card's verification credentials is passed as a document to another workflow sub-process "Proceed purchase process", which is responsible for purchase delivery to the customer. In Figure 4 we have included a small workflow example of the payment process of the insurance company, which is more in line with our earlier examples (See Figure 2).

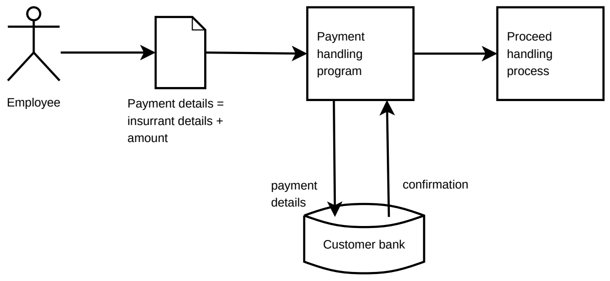

Figure 4: example: the payment process of Figure 2

3.2 Exceptions in Workflow

As much as we try to design a perfect workflow to support the processes of an organization, there will always be unexpected situations. A workflow definition (which is essentially a model) typically specifies the normal, or ideal, flow as well as possible deviations which can occur. The *Workflow Management Coalition* terminology does not include a definition of workflow exception. Therefore, many researches have come to define these unpredictable situations in their own manner. In the context of enterprise architecture, an exception is the projection of real world situations, which in some manner is of an ad-hoc origin (See Definition 1.2).

Exception are usually caused by application errors, failures or deviations, and sometimes of the lack of performance and efficiency. Exceptions mainly affect the business goals which a particular workflow helps achieve. Here again we deal with the context in which particular workflow is executed. That context usually is reason how exceptions are classified [Dellarocas and Klein, 2000]. The classification of exceptions in a business context pertains to either the goals and assumptions, are related to activities and how they are related to resources. These classes covers almost all main aspects of enterprise architecture. Furthermore, we must point out that exceptions occurs during the run-time of workflow which is why it is most difficult to deal with. One can plan and try to cater for possible deviations but there will always be unexpected behavior!

In [Aalst, 2000] a distinction is made between changes (referring to business logics and build-time) and exceptions (referring to run-time and technologies). The latter is usually a result of change which can be made in some layer of enterprise architecture and by so doing initiate an exception in other layers.

As we pointed out already, workflow is mostly related to the business layer, but workflow management system covers all layers of enterprise architecture and, thus, also plays an important role in the infrastructure and application layers. [Zongwei et al., 2000] classifies exception according to these levels and assumes that the damage of exception can be noticed in other layers than raised. And indeed it is the most important property of exception the place where it was raised. Knowing the place we can identify other exception properties, such as context, run-time input/output data and damage. In this paper we focus on exceptions which are *unexpected* (otherwise they wouldn't be exceptions, according to our definition) which have to be dealt with instantly. The important issue is to construct an enterprise architecture (model) which would actor for checking the validity and integrity of run-time data and such. Seeking to catch the exceptions of such nature we must think of services in each level which would register and define unexpected situations. Basically, an exception becomes known if it is detectable and resolvable

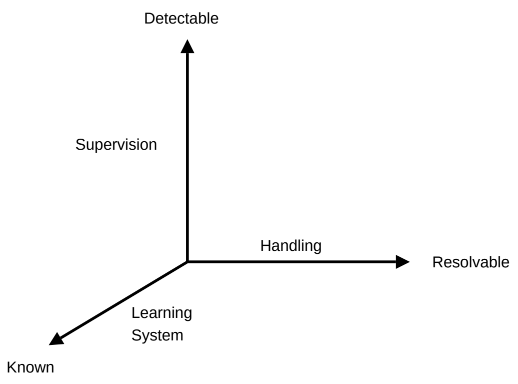

Figure 5: Three-dimensional description of exceptions.

and it becomes resolvable if it is detectable. This is illustrated in Figure 5 which represents simple three-dimensional exception specification. This specification/ classification schemes aids in dealing with exceptions [Zongwei et al., 2000].

3.3 Dealing with Exceptions in Workflow

The important question is why we should take exceptions into account so seriously and why we should try to avoid them. Should we *always* avoid them? Is the (financial) loss that results from exceptions big enough to justify the (costs of) implementing a system to catch them and deal with them automatically? There are many business cases which illustrate how unnoticed exception in workflow (which were not modeled and noticed in time) made the biggest and oldest companies bankrupt by injuring their, stable at a glance, enterprise architecture [Dellarocas and Klein, 2000]. The source of exception initiation is always human-user or a system, in both cases the seriousness off exception damage must be taken into account.

The most important step of the exception handling process is exception detection. This phase can be achieved in many ways, such as making the ideal graph of workflow and comparing every run-time step to it and if we find distinction, we register exception. The other way is to build sentinels, specific services, which would active all the time and gather information/data on workflow instance run-time and compare it to predefined ontology or taxonomy. [Aalst, 2000] always stress that the one who wants to deal with exceptions must go behind the system. The detection stage should be related to going to the back of workflow management system and gathering information there, which is usually a cause of unpredicted and unknown exception.

After identifying the right spot of exception occurrence and gathering all possible data, which will help to deal with raised situation we can pass to the second stage: the exception diagnosis stage. Where the detection phase tends to be similar for various contexts of enterprise architecture, the diagnosis phase is highly dependent on context and (therefore) differs from case to case.

This is particularly true if we care to catch not only infrastructure and application exceptions but also business logics exceptions. The best way here is taxonomies, which defines the context in detail and predefined hierarchy according to specifics of the context.

The result and actions taken to deal with the exception depends on how precise the diagnosis was made. So if detection of exception is a projection on ideal workflow model graph, then diagnosis is a method to identify the distinction of this projection and the reasons/sources why they occurred. After approval of the exception diagnosis, the last step of the cycle is to select the

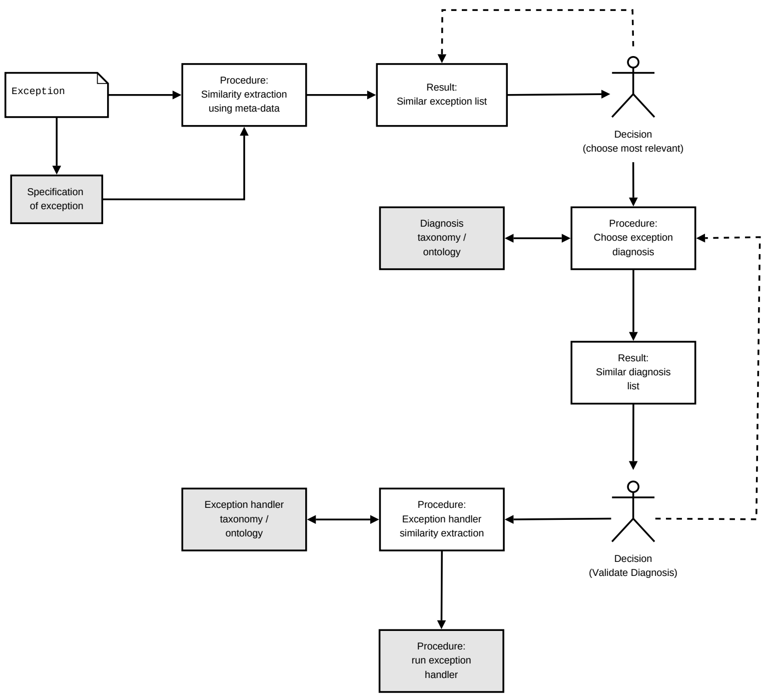
Exception
Procedure: Similarity extraction using meta-data
Result: Similar exception list
Specification of exception
Decision (choose most relevant)
Diagnosis taxonomy / ontology
Procedure: Choose exception diagnosis
Result: Similar diagnosis list
Exception handler taxonomy / ontology
Procedure: Exception handler similarity extraction
Decision (Validate Diagnosis)
Procedure: run exception handler

Figure 6: Exception handling cycle

appropriate handler. This phase is the most simple as system or human has guidelines, specification and just need to execute them correctly and finish the case. The fulfillment of business goal is not at stake during this stage though it may seem so. The decision of the impact on business goal must also be made in diagnosis phase, leaving the handling phase only to confirm the correctness of the diagnosis.

The handling phase can, none the less, be tricky as other unpredicted exceptions can be raised if diagnosis or detection phases failed. However, knowing the reasons and the cause of the exception, one only needs to select the right procedure, method, function or modification and enact it on the predefined spot, which is also identified in diagnosis phase. Note that one should not forget that the spot of exception occurrence will not always be the spot of fixing if we case about the fulfillment of business goal. The whole exception handling cycle is illustrated in Figure 6.

Exception handling as itself is a part of process re-engineering, as the running process usually is modified or changed in order to reach the goals. In so doing the exception handling procedure tends to have a danger to finish in an endless cycle of process re-engineering if any task of diagnosis, detection or handling fails.

4 Analysis

Exceptions are, obviously, perilous phenomena in business processes. They disrupt the normal process flow which may cause customers to feel untrustworthy towards the enterprise, which may cause management to be on its tip-toes which, in turn, might have serious consequences for staff who has to deal with these issues.

On a more serious note, exceptions *can* cause a lot of trouble and for the enterprise to survive they must be dealt with effectively. The first step is to re-evaluate the current (workflow) process with the methodology as described in Section 3.3. Simply put this boils down to dealing with the caused the erroneous event and make sure that the process finishes normally (or at least in such a way that the expected, proper resulted is obtained such that both clients and the enterprise benefits from it). More often than not this is a problem in itself for several reasons:

- dealing with abnormal situations costs time.
- (additional) resources are often needed to deal with the situation. However, all resources are often already allocated elsewhere.
- it may not always be clear what caused the exception to occur in the first place so it may not be apparent how to deal with it.
- In some cases the exception can point to a week link of the enterprise, which can initiate the changes in enterprise architecture itself.

Given these issues, one could argue that exceptions cost money by default, and that they should therefore be avoided whenever possible. As we have argued before, as soon as one takes a certain anomaly into account when designing or fixing a process then it seizes to be an exception. The question that remains is: would it be more costly (in the long run) to adapt ones processes each time an exception occurs or is one better off to stick to the current situation? After all, the processes were designed this way for a reason.

We argue that using architectural models may help in deciding what to do since they may help in doing an impact analysis of the two alternatives: the exception to occur again versus modification of the business process. Furthermore, using the architectural models it is easier to track down which processes, actors, information systems or hardware is involved in (dealing with) this specific exception:

- The exception always occurs in some process and, assuming the architectural models are of sufficient detail, these processes should be included in the enterprise architecture.
- By analyzing the relations of this process with other processes (i.e. by examining the business process layer in the architectural models) one may gain insight into the question: which other processes are affected by the occurrence of this exception?
- An analysis of the services that the process offers, as well as the interfaces with actors should provide an answer to the question: which actors are involved. It should not only specify if/how the customer is involved, but also which actors within the enterprise are somehow related to this event. A typical example is the absence of a certain person for a longer period of time (such as parental leave) without taking care of delegating crucial tasks of this person.
- The lower layers in the architectural models such as the application layer and technical infrastructure may provide similar insights.

When an exception occurs and is dealt with then one should *always* study the exception that occurred. Relevant questions in this respect are: Why did the exception occur? What was its impact (financially)? How likely is it that this is going to happen again? Could it have been prevented? What are the possible ways to return to the normal/expected state?

Based on the answers to these questions it is very well conceivable that it is decided that the processes should remain as they are. In other words, the exception *can* happen again. In this specific case it is advisable to keep a record of the findings related to the exception. When a similar exception occurs then such report may aid in deciding whether to act or not.

Acting would mean: modifying the business processes. In other words, the architecture of the enterprise is modified. This is, obviously, only done in rather extreme cases as modifying the enterprise architecture is quite an endeavor that requires a lot more than simply updating the architectural models!

Still, the methods of dealing with exceptions: modifying of the process and leaving as it is, can be a serious object of the discussion. Any raised exception states that the process we though is ideal can be not correct and that the enterprise architecture itself can have week links. Leaving the process as it is, is a satisfying method if we are sure, that this exception is strictly one time exception. In other cases we should apply the modification of the process method (the modification can be made through sub-processes, which is more effective, than including the fixing into the model).

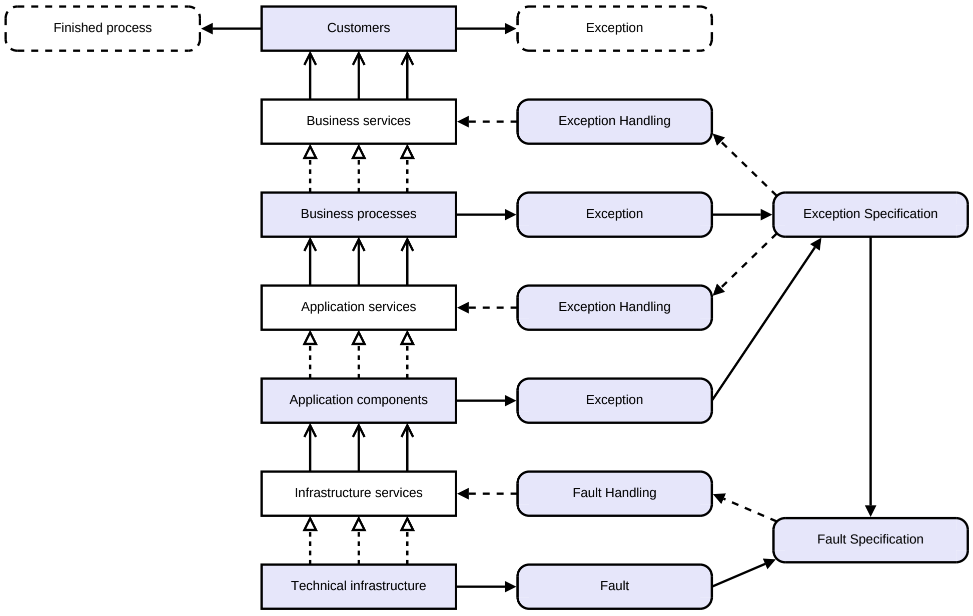

Figure 7: Layered view of architectural models with exception handling

In our view, exceptions are usually the result of the mistakes of in modeling at the architecture levelespecially given the dynamic nature of the business layer of architectural models. We therefore state that the main source of exceptions enterprise architecture is the business processes. If an exception is raised here then the exception specification must be created and merged with business services layer logic, in order to create functional exception handler.

Recall Figure 1 where we showed the three main layers of our view on enterprise architecture as well as the connection between these layers by means of services. Figure 7 extends this view with exceptions. Exceptions can occur in (components in the) main layers such as business processes, application components, technical infrastructure or even client layer. However, they are handled by the services in service layers such as business services, application services, infrastructure services. In other words, the actual architectural layers are connected by services which also have to somehow deal with their exceptions.

We propose to make a distinction between faults (which can occur only at the technical infrastructure level) and exceptions (which can occur in other layers) as the infrastructure layer acts strictly according to rules which do not involve any enterprise management logic. Still any raised fault,when specified, should be evaluated against exception description, if it does not affect any of other layers in any possible way.

In line with these observations, if an exception is raised a layer other than the business layer, its specification should be justified according to business layer in order to get the expected result. This is due to the fact that the business layer is the driver for the organization/ enterprise. As such, exceptions can be 'drilled down' to lower layers to be handled there. This is somewhat similar to the idea of push-down selection as explained in e.g., [Ullman, 1989]. This way, we can avoid the major changes in all enterprise architecture, which is costly and time receptive. Applying changes as methods through mid-layers, we will not change the layers itself, as the input and output results for layers will be same as expected.

We can return to the question/ goal of this paper: which way of dealing with exceptions is better: leaving things they way they are (assuming that is happens never again) or applying modifications the the enterprise architecture (assuming that is must be taken into serious account). The naive solution is to estimate the likelihood of the re-occurrence of the exception and doing a simple impact calculus. However, a more sophisticated solution is possible as well.

Figure 7 shows a mixed solution: In many cases it is advisable to leave the main logic (i.e., the architectural models) as is, and only apply changes to the logic that actually realize the components. For example, a component can be extended to include additional quality checks, an additional business rule can be added to create a fail-safe solution and so on. It is interesting to observe that this is entirely in line with the idea of encapsulation as used in *object orientation* (e.g., [Booch et al., 1999]) where the internal working of objects is hidden from the outside world. In our case the actual implementation of processes and components is hidden, only their services are exposed. Adopting such technique the best of two worlds can be obtained: the stability of a solid architectural description of the enterprise combined with the power of exception handling mechanisms from workflow systems.

5 Conclusion

In this paper we have studied the relation between two different views on enterprises: enterprise architecture (modeling) and workflow management. These two views are quite different in nature since the former takes a top-down view on the enterprise and the latter a bottom up view on the (processes of the) enterprise. A particularly interesting problem is the question of how to deal with exceptions that may occur.

We have presented a framework where the high-level architecture of an enterprise is divided in several logical layers. These layers are connected by services. By layering the architectural models a clear distinction can be made between business logic (as implemented by the workflow specification) and (IT) support of this business logic.

Exceptions, and particularly methods for dealing with them, has been studied extensively in workflow literature. Our main contribution is to show how exception handling related to enterprise architecture. In our view, completing a business process (and thus: the workflow) always has the highest priority; after all, customers should not be disappointed. This implies that exception handling must, if possible, be drilled down to lower levels to be dealt with.

The service concept helps us when dealing with exceptions. It is, in our opinion, keep the business architecture as stable as possible since this will result in as little changes in day to day operations as possible. Exception handling should be incorporated in the internals of the services such that the service interfaces remain stable. In other words, exception handling is encapsulated in services.

References

[Aalst, 2000] Aalst, W. v. (2000). Exterminating the dynamic change bug. a concrete approach to support workflow change. Technical Report BETA working paper series, WP51, Eindhoven University of Technology, Eindhoven, the Netherlands, EU.

[Booch et al., 1999] Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modelling Language User Guide*. Addison Wesley, Reading, Massachusetts, USA. ISBN 0201571684

[Dellarocas and Klein, 2000] Dellarocas, C. and Klein, M. (2000). A knowledge-based approach for handling exceptions in business process. *Information Technology and Management*, 1(3):155–169.

[Franken and Janssen, 1998] Franken, H. and Janssen, W. (1998). Get a grip on changing business processes: Results from the Testbed-project. *Knowledge and process management*, 5(4):208–215.

[Gils, 2002] Gils, B. v. (2002). Application of semantic matching in enterprise application integration. Master's thesis, Tilburg University, The Netherlands, EU.

[Heuvel, 2002] Heuvel, W.-J. v. d. (2002). *Integrating Modern Business Applications with Objectified Legacy Systems*. PhD thesis, Tilburg University. ISBN: 90 5668 099 4

[Hollingworth, 1995] Hollingworth, D. (1995). The workflow reference model. Technical Report TC00-1003, Workflow Management Coalition. Issue 1.1.

[Hwang et al., 1999] Hwang, S., Ho, S., and Tang, J. (1999). Mining excepton instances to facilitate workflow exception handling. In *6th International Conference on Database Systems for Advanced Applications (DASFAA '99)*, pages 45–52.

[IEEE, 2000] IEEE (2000). Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA. ISBN 0-738-12518-0
`http://www.ieee.org`

[Lankhorst, 2005] Lankhorst, M., editor (2005). *Enterprise Architecture at Work : Modelling, Communication and Analysis*. Springer, Berlin, Germany, EU. ISBN 3-5402-4371-2

[Lotus, 1999] Lotus (1999). *Domino Workflow: Automatic Real-World Business Processes*. Lotus Development Corporation. whitepaper.

[Mayer et al., 1995] Mayer, R., Menzel, C., Painter, M., deWitte, P., Blinn, T., and Perakath, B. (1995). Information integration for concurrent engineering (iice), idef3 process description capture method report. Technical report, Knowledge Based Systems, College Station, Texas, USA.

[Pastore, 2000] Pastore, M. (2000). Customer loyalty key to e-commerce profitability. http://www.clickz.com/stats/sectors/retailing/article.php/331431. last checked: 26 July 1005.

[Rijsenbrij, 2004] Rijsenbrij, D. (2004). *Architectuur in de digitale wereld (versie nulpuntdrie)*. Nijmegen Institute for Information and Computing Sciences, Radboud University Nijmegen, Nijmegen, The Netherlands, EU. In Dutch. ISBN 90-90188285-3

[Sharp, 2000] Sharp, A. (2000). *Workflow Modeling: tools for process improvement and application development*. Artech House computing library. 1-58053-021-4

[TOGAF, 2004] TOGAF (2004). *TOGAF – The Open Group Architectural Framework*. The Open Group.
`http://www.togaf.org`

[Turban et al., 1999] Turban, E., Lee, J., King, D., and Chung, H. (1999). *Electronic Commerce, a managerial perspective*. Prentice Hall, Upper Saddle River, NJ 07468, USA. ISBN: 0-13-975285-4

[Ullman, 1989] Ullman, J. (1989). *Principles of Database and Knowledge–base Systems*, volume I. Computer Science Press, Rockville, Maryland, USA. ISBN 0716781581

[UML, 2003] UML (2003). *The Unified Modeling Language*. the Object Management Group (OMG). version 1.3, formal/03-03-01.

[Vojevodina and Kulvietis, 2005] Vojevodina, D. and Kulvietis, G. (2005). Change and exceptions in business process: handling differences. In *5th WSEAS international conference on simulation, modelling and optimisation*.

[WfMC, 1999] WfMC (1999). *Workflow management coalition terminology & glossary*. Workflow Management Coalition. Document status - Issue 3.0.

[Zachman, 1987] Zachman, J. (1987). A framework for information systems architecture. *IBM Systems Journal*, 26(3).

[Zongwei et al., 2000] Zongwei, L., Sheth, A., Kochut, K., and Miller, J. (2000). Exception handling in workflow systems. *Applied Intelligence*, 13(2):125–147.