Crucial Service-Oriented Antipatterns

Jaroslav Král and Michal Žemlička
Charles University, Faculty of Mathematics and Physics
Department of Software Engineering
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
kral@ksi.mff.cuni.cz, zemlicka@ksi.mff.cuni.cz

Abstract

Service-oriented architecture is now the most popular software engineering concept. Software related antipatterns – commonly used seemingly good but in fact wrong solutions – can have adverse consequences of varying importance. It implies that the use of an antipattern can and should be viewed as a risky event. It follows that some principles of risk management can be used. We propose a method based on slightly simplified procedures of risk management and assessment. Using the procedures we give a short list of the most risky antipatterns, i.e., antipatterns occurring very often and having crucial consequences and present principles of antipattern refactoring. We discuss the following crucial antipatterns: No Legacy (development from scratch), Standardization Paralysis, Business Process for Ever (Full Automation), Sand Pile (too fine grained services), On-Line Only (No Batch Systems). The discussion of antipatterns is based on a long-term experience with service-oriented and service-oriented like (e.g., process control) systems and on the analysis of practice. Contributions of the paper: evaluation of antipatterns as risky events, specification the properties of service-oriented systems in small firms and in e-government, differences between object-oriented and service-oriented antipatterns, requirements on service interfaces, and the list of the service-oriented antipatterns being the most important ones according to the evaluation.

Keywords: antipattern, risk management, SOA type, confederations, antipattern evaluation.

1. Introduction

We will discuss the antipatterns (wrong practices) in service-oriented architecture (SOA). We say that a system has service-oriented architecture if it is a virtual peer-to-peer network of loosely related software components having properties mirroring the behavior of real-world services. Technically such systems consist of components/services and middleware. The details and effects of such system vary. We will discuss this point in details.

Typical SOA systems are formed by a ”kernel” network of services providing main system capabilities. In enterprises they are often the services supporting manufacturing like inventory control, machine floor supervision, etc. The components can be for different SOA types of different sizes. They can be large legacy systems, large third-party products, or they can be almost all quite small software components (redeveloped) from scratch. The used interfaces and communication protocols can vary. The basic communication mode is asynchronous message exchange. Message formats can be programming oriented (based on remote procedure call technique and fine grained) or user domain oriented (e.g., XML-based messages mirroring user domain languages).

Service orientation and service-oriented architecture (SOA) are the leading edge of contemporary software engineering. Service orientation is a new paradigm for business-oriented software. In the area of real-time (process control) systems, the main principles of SOA are used for decades. Business is, however, different from technology, so business-oriented SOA systems have specific aims, users, and development practices. As such they require specific good practices and turns – patterns – that can be different from the ones known from object-oriented philosophy.

The paper is organized as follows: Basic facts on antipatterns, antipatterns as risks and risk management procedures, SOA architecture types called confederations and unions, list of most risky antipatterns in unions and confederations and their solution, differences between SOA antipatterns and object-oriented antipatterns.

2. Backgrounds

The practitioners collected an impressive collection of ”SOA wrong practices” and wrong solutions called antipat-

terns (compare [5, 4]).

A pattern [8] is a solution of some problem/task known to work properly.

There can be various types or roles of patterns ([2]), e.g.:

business patterns supporting the interactions of users, business, and data,

architecture patterns supporting the construction of a software architecture – for example integration of several business patterns;

application patterns used to implement interaction of application components and data in business patterns or architecture pattern.

An antipattern is according to [5] a seemingly good solution that is commonly (repeatedly) used but often failing to provide satisfactory results. The specification/description of an antipattern should specify why the antipattern looks to be a recommendable solution. To describe an antipattern it is recommended to specify its symptoms, root causes of the antipattern and consequences of the use of the antipattern [5]. The crucial part of the antipattern description is its (refactored) solution – a way of modifying the antipattern to avoid its wrong consequences; i.e., how to convert it to a good solution.

According to [5] we have in object-oriented environment Software Development Antipatterns, Software Architecture Antipatterns, and Software Project Management Antipatterns. We will see that SOA antipatterns can be of similar types but with different consequences. Some object-oriented antipatterns are patterns in SOA and vice versa (compare the object-oriented antipattern Islands of Automation; it is a crucial SOA pattern). Some SOA solutions solve issues of several problem domains at once (e.g., software development and software management)[1]. The relation between SOA systems and business is tighter than in the object-oriented systems. It leads to yet SOA-specific antipattern: Specification Antipattern referring to wrong attitudes used during the requirements specification phase. Under these circumstances the user involvement in requirement specification as well as in system development [17] is crucial. Users as well as IT professionals should therefore apply some attitudes of agile business and agile software development.

The SOA-related research and experience with SOA in practices produced lists of SOA-related antipatterns [4, 12, 24, 6]. The lists do not attempt to depict the importance of individual antipatterns: how often they occur and how critical losses they cause. Such an evaluation depends, however, on the variant of SOA in which the antipattern is applied. We attempt to evaluate these issues applying the principles of risk management.

[1] All these facts indicate that the service-oriented philosophy is substantially different from the object-oriented one. It can be one of the reasons why SOA is not easy to apply although it seems to be intuitively clear.

2.1. Service-Oriented Software Systems

SOA in our understanding is a virtual peer-to-peer network of software entities called (software) services having many properties common with real-world services. Such systems are formed by the services and a middleware enabling asynchronous communication between the services. The capabilities provided by the middleware vary depending on different conditions. The middleware can include Enterprise Service Bus (ESB, [7]) but the use ESB can be sometimes contraproductive. This issue will be discussed below.

Technically are the services implemented as permanently active service processes communicating asynchronously (batch services are in this case permanently active but having a long latency). We do not exclude systems based on the tools like MessageQueue (MQ) and its descendants. Our concept of service-oriented systems is broader than the one proposed by large software vendors. It is, however, appropriate for many systems supporting, e.g., small-to-medium enterprises or e-government. So the extension of SOA concept we propose is appropriate for many (if not the majority of) systems occurring in practice. For example the integration of existing software items is of the highest priority. It is often denounced not to be good solution. We must understand that the broader treatment of service orientation and service-oriented architecture implies substantially different properties of the resulting systems.

According [15] we can recognize two basic types of service-oriented systems: confederations and alliances. Alliances are collections of components able to search or ask for their cooperating partners. Typical alliances are e-business systems based on web services in the sense of W3C. Confederations are systems where individual components are aware of their cooperating partners. Their communication need not be based on international standards. Typical examples are e-government, information systems supporting global enterprises, or health care systems or some process control (soft real-time) systems.

3. Evaluation of Antipatterns and Risk Management

The research and study of the antipatterns collected a list of (possible) antipatterns occurring in practice. There are risks of loses related to (caused by) each antipattern. From the management's point of view an application of an antipattern usually (with some probability) leads to a project failure. It is therefore a risk. So it is meaningful to apply (adapt) some techniques of risk management [10].

P \ L	low	high
low	low	high
high	low	very high

Table 1. Evaluation of E

According to [10] the risk management consists of the following stages:

1. Risk identification. The construction of the list of antipatterns.
2. Risk assessment. Estimate the expected (i.e., mean) risk loss E(i.e., risk related to the application of an antipattern). It is recommended to estimate E as a product $E = p \cdot L$ where p is the probability that the antipattern will cause the loss L. As we have only rough estimates of p and L, we can use fuzzy estimates (we use the experience of a software developer [27]). It usually suffices to estimate p as *low* and *high* and L as *low*, *high* and set E = *very high* for p = *high* and L = *high*; E = *high* for L = *high* and p = *low*; and E = *low* in other cases. So we have three classes of antipatterns with E = *low*, *high*, and *very high*, see Tab. 1.
3. Risk ordering. We order the antipatterns according the expected loss E. We will discuss the antipatterns having the largest E. It is therefore meaningful to look for the solution of the antipattern having the assessment *very high* (we call such antipatterns critical) and sometimes for the antipatterns with assessment *high*. The assessment should be the part of the description of the antipatterns in a given environment. We shall discuss mainly the critical antipatterns in confederations. Many conclusions hold for all service-oriented systems.

The above assessment can be felt to be rough. In this case we can use for p the degrees *low*, *rather low*, *rather high*, *high*. If necessary, we can extend the scale further. The experience shows that in situations we are discussing the rough assessment is better than the fine one [28]. Sometimes it is good enough to estimate only the level of E directly – without referring to p.

4. What SOA for What Purpose

Simple/small process control systems (real-time systems) were the first systems applying the crucial principles of SOA (see, e.g., [13]). Components (services) were software components driving/supervising rather intelligent devices of real world. Typical example were operation systems of minicomputers and systems controlling manufacturing like flexible manufacturing systems [13] or computer integrated manufacturing (CIM).

The middleware (transport tier) were mainly supported by tools of operating systems like mailbox, the system need not be distributed. The number of components was small and the components were known so the communication protocols and message formats can be agreed. The components were usually written from scratch and they as a rule used interfaces based on a variant of remote procedure call format. The format is "programmer oriented" – i.e., designed mainly to cover the needs of developers.

To summarize – real-time systems have the following features:

- small components developed form scratch,
- mainly known components (we call such SOA *confederations*),
- simple middleware using communication protocols, especially message formats, that need not be user oriented as they are not used by users,
- interfaces tend to be fine grained, procedural, developer oriented,
- not too opened, limited reuse.

Note that communication partners need not be looked for at the start of their dialog.

4.1. Alliances

In e-commerce the communication partners can be looked for all over the world. The implementation of such systems is usually based on web services. It follows that world-wide networks and open standards must be used and the interfaces are difficult to be adapted to specific needs of a given system – it is especially a difficult problem in the situation when immature standards only are available.

Such systems have the following features:

- The communication partners must be looked for at the beginning of communication, possibly all over the world,
- Internet is usually used as (the kernel of) middleware and web services are a good solution,
- almost all the aspects of implementation is based on open standards,
- highly open systems,
- quite frequent use by different users, reuse possible.

We call such systems *alliances* for short [15]. Alliances are typically used in *e*-commerce.

4.2. Confederations

The majority of service oriented systems contain a virtual kernel being a subnetwork of software services providing the basic capabilities of the whole system. The network contains a limited number of well known services. The communication protocols of the services can be agreed. We call such systems *confederations*. Examples are *e*-government, information systems of municipalities, organizations with professional bureaucracy, ERP systems, etc.

The development of confederations strongly depends on the fact whether the service requirement process is supervised by a quite strong central authority or not. The first case is typical for the ERP systems of large enterprises, especially if they have division structure and machine bureaucracy [22].

In the second case the services in the kernel are almost independent, like the states in the European Union. We call therefore such confederations *unions*. It follows from the above list, that software unions occur quite often. They are even typical for the ERP of small firms being often induced to integrate various legacies and third party products.

4.3. Unions

It is crucial to decide what SOA type is to be developed. We can distinguish the following cases:

- *Alliances.* Highly open systems where communicating services must be looked for. They are typical for $e-$commerce.
- *Confederations.* Systems where communicating services in the system kernel knows each other. It is typical for a wide spectrum of SOA systems. There are the following main variants of confederations (Figure 1):
  - *Soft real-time systems.* Almost closed systems with quite small components and typically fine grained interfaces. The interfaces are not intended to be used by users. It is typical for some soft real-time process control systems.
  - *Enterprise confederations.* Semi-open systems supporting large enterprises having machine bureaucracy [23, 22]. The enterprise has enough resources to develop or buy the whole system at once. Architecture is typical for the information systems known as Enterprise Resource Planning (ERP) of global enterprises. Typical is the use of Enterprise Service Bus (ESB, [7]). The core of the confederation is so large that it is meaningful or necessary to use ESB.
  - *Unions.* Almost open SOA systems. Their kernels are built of a quite small number of almost independent services knowing each other. It is the SOA variant typical for the information systems of the large organizations with professional bureaucracy (e-government, schools, etc.) or small or medium-sized enterprises (SME). SME have organization near to ad-hoc-cracy.

The SOA patterns and antipatterns are different for different SOA types. We mainly will discuss the case of unions. Unions rarely use ESB as they integrate a quite small number of applications or systems. Compare the systems of particular offices of a local administration. Unions are typical not only for system having professional bureaucracy but also for also for small and medium-sized enterprises as these enterprises usually do not have enough resources to rebuild their systems completely. Their organization is moreover specific.

In e-government, enterprises (compare [21, 18]), health care systems, etc. the resulting system is built from legacies, third-party products and newly developed systems. It is preferable to wrap the systems such that they have properties mirroring the behavior of real-world services, e.g., asynchronous communication protocols and user-oriented (usable) interfaces. The middleware can and often must use Enterprise Service Bus [7]. It is good when services like data stores and service adapters are used. We call such components *architecture services*.

Unions are formed by a core network of complex application services (mainly legacy systems and third-party products), architecture services, and a middleware. The number of application services is small and the services are known. So their interfaces with the help of architecture services can be agreed – such systems are therefore confederations with large application components connected with help of architecture services and middleware. Some services can communicate with the ”peripheral” or ”foreign” services using the principles used in alliances.

Features of unions in practice:

- large permanently used components that can be integrated into SOA together with their local interfaces (e.g., client tiers) without putting them out of operation for a longer time;
- sophisticated middleware enhanced by architecture services;
- quite large application services – often reused legacy and third-party systems;
- user-oriented coarse-grained interfaces of application services;
- based mainly on the use of open standards but some solutions or their parts (e.g., interfaces) can be propri-

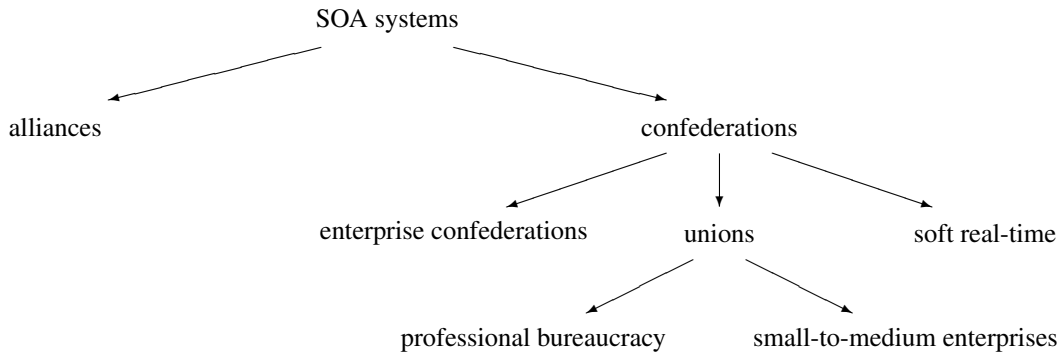

Figure 1. Hierarchy of SOA systems

etary, if appropriate; it is the case when we need the user-oriented declarative coarse-grained interfaces.

- the application services are almost independent (like states in the European Union).

Unions can be used in the development and maintenance of large systems and they can also effectively support the modernization of information systems for small-to-medium enterprises (SME). We will discuss mainly the unions. The reason for it is that we have enough experience with unions and that unions occur frequently. We believe that below given antipattern list is common for all confederations. The evaluation can be different for some antipatterns. Possible candidates for it are "No Legacy", and "Standardization Paralysis".

The situation in alliances is different and will be a topic of further research. In alliances it is more necessary to use open standards, agile business processes are therefore more difficult to apply.

5. Architecture Antipattern: Mis-Selection of SOA Type

SOA is in fact a concept covering multiple technologies. Improper selection of SOA type is therefore an improper selection of a technology to be used.

Symptoms and Consequences

The fact that there are several SOA types, is often ignored. In practice semi-SOA systems are frequently used by wrapping constituent software entities using various message queuing tools like MQ or even low-level tools provided by operating systems.

The different SOA types have different patterns and antipatterns. The proper solutions for different SOA types differ as well. The improper choice or missing choice of proper SOA type leads as a rule to project failure or to substantial losses.

The antipattern is "applied" already in the vision and requirement specification phases – if the requirements are hard to be mapped to possible solutions, or if the used technology restricts the expected features of the system, it is likely that an improper SOA type has been chosen. It can be induced by the application of improper standards or by the marketing of software vendors The typical consequence is failed or too effort consuming a poorly maintainable project often also missing some of the project requirements.

Assessment

The fact that there are several significantly different SOA types is quite unknown. The needed knowledge is blocked by strict standardization effort and by the marketing of large software vendors. p is therefore *high*. The consequences – completely failed project or project fulfilling the goal only partially mean very high loss. L is therefore also *high* to *very high*. Hence E is o *very high*.

Solution

To prevent this antipattern it is good to make the requirements specification precisely and without preselecting the solution type. When the antipattern is already detected, it is necessary to return to the requirements specification and analysis and try to map the requirements to potential solutions. Typical tasks and solutions are discussed below.

Note

It is crucial that SOA in our understanding is any system being a virtual peer-to-peer network of software artifacts behaving like real-world services. This concept is broader than the concepts adopted in SOA related standards, com-

pare the standards by OASIS or W3C or the concept supported by large software vendors.

Too narrow definitions can limit the use of SOA principles and benefit from SOA advantages. It may be one reason of the falling popularity of SOA in the last year, compare the study of Gartner Group from January 2008.

In practice we must integrate batch system, build SOA in bottom up manner etc. It is not difficult to implement it we properly wrap integrated applications and eventually treat them as services communicating in bulk mode and having a great latency. We can occasionally integrate batch applications wrapped by tools like MQ to behave like services.

The techniques developed for service oriented systems can often be used outside SOA. Missing to use it is itself an antipattern. The solution of the discussed antipattern facilitates or implies the solution of several known antipatterns. Examples are: "SOA = Web Services" and "Big Bang" antipatterns.

6. Methodological Antipattern: Fine-Grained Interfaces

Fine-grained interfaces cannot mirror the coarse-grained interfaces of real-world services properly. Fine-grained interfaces have substantial technical drawbacks.

Symptoms and Consequences

People trained to design object-oriented code as a large structured collection of methods and classes being in fact a big collection of procedures each doing almost nothing. It leads in service-oriented framework to the development of systems consisting of services having fine-grained interfaces and being themselves fine-grained. This antipattern was firstly described in [4]. The corresponding communication protocols usually use formats based on the remote procedure call (RPC). Such an attitude is induced also by the fact that RPC is a straightforward way to "reuse" the fine-grained interfaces of software components being in fact wrapped components based on object-oriented methodology.

It all together leads to services being "talkative" and equipped by interfaces that are not user-oriented. The first property leads to the overloading of communication links, the second in fact implies a difficult development of agile business processes, a lot of user discomfort, and a more complex implementation of some management operations like insourcing and outsourcing.

Fine-grained interfaces tend to disclose too much on the technical details of the services. It implies that the interfaces are too much influenced by the changes of the interiors of the services. It is generally known to be an unpleasant property.

Assessment

The antipattern is quite common for all SOA types. p is therefore *high* for all SOA types. The losses L are quite large for unions, especially for the unions for SME. In large enterprises having enough resources to decrease L using appropriate means the problems is not so severe.

For unions p is *high*, L is *low – high*. E is therefore *high – very high*.

For enterprise confederations is L quite *low*. E is in this case *low – high*.

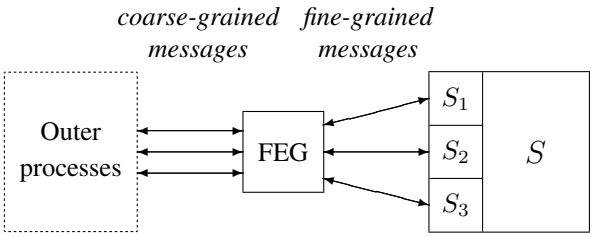

Figure 2. Refactored antipattern Fine-Grained Interfaces

Solution

We can use sophisticated forms of enterprise service bus connectors in the case of enterprise confederations. The second, and may be better, solution is the use of specific architecture services acting as service adaptors. Such services in [20] called *front-end gates* (FEG).

FEG transform n-tuples of fine-grained messages into declarative complex messages having rich semantics, and vice versa. The solution with FEG is applicable for unions as well as for enterprise confederations.

7. Specification and Architecture Antipattern "No Legacy"

Insufficient reuse of legacy systems and third-party products limits substantially the benefits of SOA – especially for small and middle-sized enterprises or for e-government.

Symptoms and Consequences

It is often required that the developed system should not contain any "obsolete" parts – e.g., legacy systems. Compare the antipattern "Lava Flow" [5] known from object-oriented methodology. The reuse of existing software is, however, the most important opportunity of service orientation. In the service-oriented setting it is a very costly antipattern as the main advantage of service orientation is that

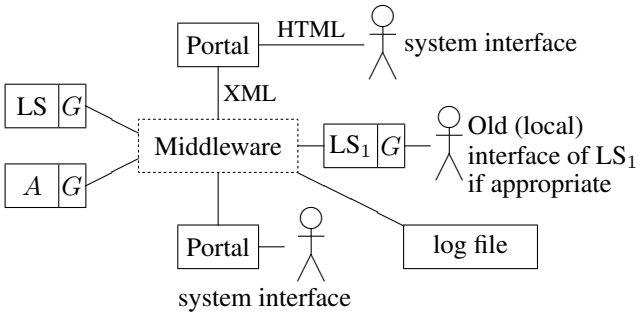

Figure 3. Union integrating legacy systems L-S and LS_1.

it enables the integration of autonomous systems, especially legacy systems. Throwing out of legacy systems causes superfluous immense additional investments into development and implementation of new systems. Very large systems would not be then implementable at all. In other cases it can cause unnecessary testing expenses and retraining of end users.

Assessment

This antipattern occurs very often. The reason is that in the object-oriented world this antipattern is a strongly recommended pattern, compare the object-oriented antipatterns like Island of Automation. Software vendors are not happy that they should integrate and guarantee foreign software. So for unions p is *high* and L is *very high*. The assessment of the antipattern is for unions therefore *very high*.

The consequences of the antipattern bearable for software confederations. In this case p is *high*, L is *high* and E is therefore only *high*.

Solution

At first the project management should accept that it is good to leave some older parts (legacy systems) in a new system as the old parts have useful capabilities and can be very stable. Then we should choose the candidates for integration. The candidates must be then equipped with user-oriented interfaces (portals) to be used by their communication partners. Let us repeat that user-oriented interfaces are based on formalized text straightforwardly mirroring the languages of the users' world. We can use front-end gates [14] to equip the legacy systems with user-oriented interfaces. User-oriented interfaces enables a very powerful implementation of the software engineering principle of information hiding.

If designed carefully, the integration of the systems does not imply any change of "existing" or "old" interfaces of the legacy system LS (Figure 3). Moreover, the local users need not lose their feeling of ownership to "their" system and can bear the responsibility for its data and functions. There can be some political (feeling of power and influence) and organizational (autonomy of divisions/departments or offices in e-government) reasons.

Service orientation is the best way of refactorization of Stovepipe Systems and Stovepipe Factory antipatterns (see [5]). Well working legacy systems simplify outsourcing and reduce the necessity to retrain the end users to work with the new system. Last but not least the use of wrapped legacy systems enables large investment savings – well behaving legacy systems need not be redeveloped, users do less errors and need not be retrained. Note that FEG and portals are from technical or development point of view similar.

Service orientation is the best known way of supporting decentralization of enterprises.

A proper solution of the "No Legacy" antipattern is a precondition of the solution of "Big Bang" antipattern and enables a smooth application of incremental development.

Notes

This antipattern is often a consequence of the antipattern "All From Scratch" and can be also a consequence of the SOA-variant "Vendor Lock-In" antipattern known also from the object-oriented world. In fact the antipattern "No Legacy" is the SOA-variant of the object-oriented antipattern "Reinvent the Wheel" [5].

We have experienced several projects failing due the application of the antipattern "No Legacy". The proposed solution of the antipattern is typically blocked by the antipattern "Standardization Paralysis" attempting to apply cumbersome, complex, and often immature standards everywhere.

8. Management and Design Antipattern "Standardization Paralysis"

The overuse of many (especially immature) standards is contraproductive and can cause some known antipatterns like "Vendor Lock-In" or "Technology Bandwagon".

Symptoms and Consequences

XML enables an easy development of languages transformable very quickly into standards. Many people believe that the standardization based on (quickly changing and obsoleting) standards is a proper solution. The result is that the developers often strongly depend on software vendors and that any solution based on legacies is therefore almost impossible. These effects can be according to [12] known as Technology Altar Antipattern. It often leads to SOA antipatterns like "SOA = Web Services" [4].

The psychological roots of the antipattern has a lot of common with the object-oriented antipatterns "Continuous Obsolence" and "Lava Flow" [5]. An example of this antipattern is the e-government of one country where all the newly incorporated systems must be certified for compatibility with a very quickly changing set of interface rules. It blocks the e-government development.

Assessment

The requirement to standardize all system details is very strong and common. So p is *high*. An improper standardization can induce the use of the antipattern "No Legacy" and limit agile methods of development. It implies the dependency on the product of large software vendors. It implies something like the object-oriented antipattern "Vendor Lock-In". It in fact creates an unnecessary initial barrier for the application of SOA for unions. This consequence is more important for unions where L is therefore *high* to *very high* and E is therefore *high* to *very high* too.

For other SOA types E is *low*.

Solution

In the case of unions and probably in other types of service-oriented systems as well we can at first specify local (private) interface standards, test them for usefulness and other quality aspects in practical applications. Then we can lately transform them into public (ISO) standards after they are available.

9. Specification Antipattern "Business Process for Ever"

It is often believed that the processes are defined so well that no more changes will be necessary. It is in long-term not true even for large enterprises. For smaller companies or institutions in dynamic environment it is not true in almost always.

Symptoms and Consequences

The antipattern can be also called "No Businessmen Involvement" or "Full Automation". It is based on the conviction that well-defined business processes should not be modified easily, if ever. The changes of the process can be implemented by a highly specialized people or teams. It, however, assumes that there are business data of a good quality (i.e., accurate enough, accessible, timely, trustable, not changing quickly, in volumes that are large enough, etc. – compare [29]).

Such assumptions are correct for stable business environments and for large firms having repeatable business processes not too influenced by globalization. Such enterprises have enough resources and enough data to develop processes of a very good quality. It definitely does not hold for small- and medium-sized firms especially in small economies. The books like [9] on the theory of constraints indicate that the assumptions need not hold even for large firms as the business process philosophy must be substantially changed quite often.

If, however, a businessman responsible for the process cannot change the process structure, then he/she cannot be responsible for business consequences of the process. The businessmen cannot even commit business steps if he/she does not know all relevant business information – e.g., trustability of some data that were available to process designer. The situation can be characterized as "fully computerized business processes – no agile user involvement allowed". Such a solution is often unacceptable. Adverse properties of fixed (non-modifiable or modifiable with difficulties) business processes are the following:

1. We can require almost no responsibility of process users (business process owners) for the business process consequences. They can act only as observers and wrong in long term perspectives. It can be fatal in emergency situations.
2. In small and medium firms the data and information are not good enough to enable the definition of stable business processes. This issue is often important for large firms too.
3. The business environment changes, and, e.g., due globalization, the art of business is changing.
4. The detailed specification of business processes is very expensive and time consuming. It cannot be implemented on-line.
5. It is good to train people to cope with unexpected events or changing conditions. We can therefore conclude that it can be often good to allow the users:
   (a) to supervise business processes,
   (b) to commit the business steps if necessary,
   (c) to redefine/change on the fly (in agile style) some parts of the business processes,
   (d) to save/remember the changes of the processes as a part of business intelligence,
   (e) to develop the process from scratch via logging process owner commands.

   Note that a) provides the tools allowing customers and other processes to observe the progress of the process. Similar requirement can be found in [25].

Assessment

There is a widespread conviction that business processes are too important and too difficult to develop to allow business people to change them. The probability of the "use" of the antipattern is therefore *high*. The business consequences tend to be *very high* for unions as agility is there highly desirable. So the assessment of the antipattern is for unions *very high*, i.e., extreme. The antipattern Business Process for Ever is therefore in business environment critical.

The consequences for enterprise confederations are usually bearable (L is *low* to *high*) and as the processes are well designed, p is *low*. We have therefore for this case $E = low$.

Risk evaluation of this antipattern for other SOA types requires further research.

Solution

Any refactoring of the antipattern should enable the on-line user involvement of the user into process execution (it is the responsible user must be able to supervision and on-line modify its processes). The service is defined (e.g., in BPEL [3] or in Aris [11] or UML Action Diagram notation [26]) as a network of actions (operations) provided by software services (we shall call them *application services*). In order to enable the user involvement in business processes it is practically necessary to make application service interfaces user oriented. It is to make the interfaces to mirror user domain professional language, and to mirror (if possible) requests of real-world services (e.g., generate an invoice, bill, consignment/remittance). From the point of view of developer such interfaces are rather declarative (i.e., saying what to do rather than how to do it) and coarse grained.

Using pattern called in [18] *Front-End Gate* (FEG) the interfaces not being user oriented can be usually transformed into user-oriented ones and vice versa.

FEG of an application service S is technically a peer in a peer-to-peer network. Logically it accepts sequences of possibly fine-grained messages from S and transforms them into coarse-grained messages for communication partners of S and vice versa.

The use of FEG can solve the antipattern Chatty Service [4] the assessment of which is *high* to *very high* as L for it is not fatal. FEG can avoid the necessity to use the messages being too developer oriented. It can be called Cyberspace Antipattern. This antipattern usually implies that the service interfaces are not user oriented and it implies the antipattern Business Process for Ever.

The Cyberspace Antipattern occurs quite often, so it should be assessed as critical. FEG can be used to solve antipattern Grey Services when interfaces disclose some implementation details (i.e., services are not used as absolutely black boxes).

Note that user-oriented interfaces simplify the development as they enable the development of powerful screen service prototypes with almost no additional effort [16, 19].

The business services should be implemented such that the implementation fulfills all the conditions a) through e) and additionally enabling the use of business control/modeling data of different types can be based on the pattern Process Manager.

Process Manager

Business processes must admit on-line involvement and supervision of process owners into their execution. According [18] the reasons are:

1. The process model/definition is based on data that need not be timely, accurate, or complete.
2. The business conditions changed or some conditions are not valid any more.
3. The process owner can be obliged to agree with some risky process steps.
4. The information on the process should be understandable for experts (not necessarily IT ones), e.g., at trial.
5. The process model M should be stored as a part of business intelligence.
6. It is desirable to be allowed to have process models in different languages, e.g., in BPEL [3], Aris, [11], workflow [30], or in a semistructured text. The reason is that business process models can be as a part of business intelligence collected during a long time. It is, they must be able to take into account the whole collected experience, e.g., various business documents like manufacturing logs, old business process description documents based on different methodologies. The requirement is especially important for SME where it must be even possible to enable process owners to control business processes having no definition at all or a very informal one. It in fact enables the use of otherwise blocked knowledge of process owners.

As it is not desirable to have much centralized services in peer-to-peer systems (compare experience with UDDI [4]) we can use the following hints:

1. When a process is enacted (typically on the request of its owner O), generate a new service P called *Process Manager*. During the generation a process model M (if any) is transformed into a process control data C parametrizing M using parameters provided by O. O can generate C directly without M, if appropriate. M can be copied from a data store.

2. The process P during its execution generates using C service calls. The calls can be synchronous (call and wait for answer) or asynchronous (just send a message). C can be on-line modified.

3. It is important that if the process owner can supervise the process run and the process run is understandable by non IT experts then the services should have interface based on the languages of user knowledge domains – we say that such interfaces are user oriented. User orientation implies some limitations in the use of classical web services in service-oriented systems. User-oriented interfaces have many software engineering advantages – stability, conciseness, ability to hide implementation details. They enhance reusability of the services. The user-oriented messages are semantically rich, so the communication channels are less loaded.

Using Process Managers to handle business processes is not only refactoring of the antipattern "Business Process for Ever", it is even itself a pattern [20].

10. Design Antipattern "Sand Pile"

The orchestration of lots of small services is sometimes a hopeless issue.

Symptoms and Consequences

This antipattern is also known as "Fine-Grained Services". A frequent implementation of SOA is the technique "one elementary service (e.g., pay-a-bill) per one software component". The result is a large number of small components sharing common data stores. It causes inefficiencies and big maintenance problems. Similar properties are by the antipattern Atomic Services [24] but we think that the crucial problem is a wrong grouping of "atomic" capabilities. It is like to have many highly specialized car service workshops working separately – it is no enterprise providing all repairs at one spot.

Assessment

The antipattern occurs rarely for enterprise confederations as well as for unions (p is *low*) but its consequences are fatal (L = *very high*). So the assessment is in both cases *high*.

Solution

Group related "elementary" services into a composite service with common interface provided by an architecture

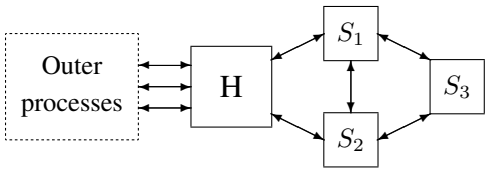

Figure 4. Refactored antipattern Sand Pile

service H called *head of composite service*. For example the composite service can be a collection of services supporting activities of a department. To be more specific: let $S = \{S_1, ..., S_n\}$ be the collection of the services supporting the department. It is required that all the messages address S or from S must pass through H (see Figure 4 and [20]).

11. Antipattern "On-Line Only"

Symptoms and Consequences

Practice indicates that there are frequent situations when some parts of the system must or should be run in batch mode. The main reasons are:

- Some legacy systems are batch systems.
- Some activities have a long latency (as they are computationally complex) or need real-world (e.g., user) responses.
- There can be software engineering reasons to use batch systems (reduction of development effort, security, etc.).

So the integration of batch systems is necessary. The integration can be via data stores implemented as services. There is a prejudice that it is an obsolete technique. The antipattern results into expensive and unstable solutions.

The importance of the combination of batch and on-line applications is discussed in a case study of a flexible manufacturing system [18].

Assessment

The antipattern occurs not too frequently as the integration of batch systems are not frequently used but consequences of it can be very high. So p is *low*, L is *very high*. The assessment E is therefore *high* to *very high* for all confederation types. Note, however, that with growing size of information systems the frequency of the cases when batch systems must be used will grow. Although the pattern may occur also in alliances, risk evaluation of this case has not been done yet.

Solution

It is sometimes good to implement a classical data store but wrap it as a service communicating in bulk mode with batch services being wrapped batch systems. The data store is the filled by batch subsystems in batches and used interactively by other services of the system [18] or vice versa.

The data store can be also used to support the enhancement of communication protocol. In this case the data store contains messages. We call such a data store *message data store service* (MDS). Sometimes a file transfer communication can be used instead data store services.

Note

MDS can be used as a service implementing sophisticated variants of inter-service communication [18]. It can be used for the resolving of, e.g., the antipattern Point-to-Point Service [4] or to implement a very sophisticated communication schemas being more complex than, e.g., publish-subscribe protocol. Data stores and MDS substantially increase the flexibility of SOA using them.

12. Conclusion

The main goal of the research of SOA antipatterns is often aimed on the extension of the list of known antipatterns. The importance of individual antipatterns is seldom discussed.

We have shown that the evaluation the antipatterns has many common features with the stage of risk identification in standard processes of risk management. In fact, there is a risk related to every antipattern. So we should apply consequent stages of risk management for the assessment of antipatterns in order to select and manage the most important risks (antipatterns).

The techniques we have used are very useful as they help to find and properly evaluate the antipatterns being the most important ones from the point of view of risk management. The most critical (meta)antipattern is Mis-Selection of SOA Type.

The further most critical antipatterns are ”Software Processes for Ever” not allowing agile business processes, ”No Legacy”, and ”Standardization Paralysis”. These antipatterns have many links to other SOA antipatterns and even to the antipatterns known from the object-oriented world. In the future we will assess risks of the antipatterns known from the lists mentioned in the references.

Note that the most critical SOA antipatterns are the antipatterns not occurring among the object-oriented antipatterns ([5]). The examples are: ”Mis-Selection of SOA Type” and ”Standardization Paralysis”. Some SOA antipatterns are patterns in object-oriented philosophy (”No Legacy”), and vice versa (”Fine-Grained Interfaces”). The assessment of service-oriented antipatterns has different results for different SOA types. We have discussed some cases of it.

The overall structure of SOA is mainly implied by communication disciplines. They are almost not controlled by any explicit tools, so their use is only the matter of attitude. It substantially increases the flexibility of the SOA systems. If, however, not used properly, it makes the system maintenance a hopeless issue.

Note that the marketing of service-oriented standards have resulted into the situation when there is, according to our meaning, a wrong conviction that SOA requires substantial initial effort and investments. A less dogmatic attitude can allow building systems having no SOA in the strict sense of OASIS and W3C but offering substantial amount of benefits typical for SOA.

It is open whether the differences of unions in small-to-medium enterprises and in large organizations with professional bureaucracy like e-government are not more fundamental than we assumed up to now. It is a very interesting topic for further research.

We applied our evaluation process of antipatterns on the atnipatterns listed in [4, 12]. The results were the following: The evaluation were *low* except the cases when the antipatterns were special instances of the antipatterns from our list. It is important that the majority of the evaluated antipatterns cannot take place or can be solved easily provided that all the antipatterns from our list are solved properly.

Acknowledgement This research was partially supported by the Program ”Information Society” under project 1ET100300517 and by the Grant Agency of Czech Republic under project 201/09/0983.

References

[1] J. Král and M. Žemlička. The most important service-oriented antipatterns. In *International Conference on Software Engineering Advances (ICSEA'07)*, page 29, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[2] J. Adams, S. Koushik, G. Vasudeva, and G. Galambos. *Patterns for e-Business: A Strategy for Reuse*. MC Press, 2001.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Specification: Business process execution language for web services version 1.1, 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/ 2009-05-14.

[4] J. Ang, L. Cherbakov, and M. Ibrahim. SOA antipatterns, Nov. 2005. http://www-128.ibm.com /developerworks/webservices/library/ws-antipatterns/ 2009-05-14.

[5] W. J. Brown, R. C. Malveau, H. W. S. McCormick, III, and T. J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, New York, 1998.

[6] S. Carter. The top five SOA don'ts, Mar. 2007. http://www.ebizq.net/topics/soa/features/7780.html?related 2009-05-14.

[7] D. A. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, 1993.

[9] E. M. Goldratt. *Critical Chain*. North River Press, Great Barrington, MA, 1997.

[10] E. M. Hall. *Managing Risks, Methods for Software Systems Development*. Addison Wesley Longman, Reading, MA, USA, 1998.

[11] IDS Scheer. Aris process platform.

[12] S. Jones. SOA anti-patterns, 2006. http://www.infoq.com /articles/SOA-anti-patterns 2009-05-14.

[13] J. Král and J. Demner. Towards reliable real time software. In *Proceedings of IFIP Conference Construction of Quality Software*, pages 1–12, North Holland, 1979.

[14] J. Král and M. Žemlička. Component types in software confederations. In M. H. Hamza, editor, *Applied Informatics*, pages 125–130, Anaheim, 2002. ACTA Press.

[15] J. Král and M. Žemlička. Software confederations and alliances. In *CAiSE'03 Forum: Information Systems for a Connected Society*, Maribor, Slovenia, 2003. University of Maribor Press.

[16] J. Král and M. Žemlička. Service orientation and the quality indicators for software services. In R. Trappl, editor, *Cybernetics and Systems*, volume 2, pages 434–439, Vienna, Austria, 2004. Austrian Society for Cybernetic Studies.

[17] J. Král and M. Žemlička. Systemic of human involvement in information systems. Technical Report 2, Charles University, Faculty of Mathematics and Physics, Department of Software Engineering, Prague, Czech Republic, Feb. 2004.

[18] J. Král and M. Žemlička. Implementation of business processes in service-oriented systems. In *Proceedings of 2005 IEEE International Conference on Services Computing*, volume II, pages 115–122, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[19] J. Král and M. Žemlička. Software architecture for evolving environment. In K. Kontogiannis, Y. Zou, and M. D. Penta, editors, *Software Technology and Engineering Practice*, pages 49–58, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[20] J. Král and M. Žemlička. Crucial patterns in service-oriented architecture. In *Proceedings of ICDT 2007 Conference*, page 24, Los Alamitos, CA, USA, 2007. IEEE CS Press.

[21] J. Král and M. Žemlička. Software for small-to-medium enterprises. In M. M. Cruz-Cunha, editor, *Enterprise Information Systems for Business Integration in SMEs: Technological, Organizational and Social Dimensions*. IGI Global, 2009. To appear.

[22] H. Mintzberg. *Mintzberg on Management*. Free Press, 1989.

[23] H. Mintzberg. *Structure in Fives: Designing Effective Organizations*. Prentice Hall, 1992.

[24] T. Modi. SOA antipatterns, Aug. 2006. http://www.ebizq.net /hot_topics/soa/features/7238.html 2009-05-14.

[25] OASIS. Asynchronous service access protocol (ASAP). http://www.oasis-open.org/committees/download.php /14210/wd-asap-spec-02e.doc 2009-05-14.

[26] OMG. Unified modeling language, 2001. Available at http://www.omg.org/technology/documents/formal/uml.htm.

[27] Z. Staníček. Private communication, 2007.

[28] Z. Staníček and J. Hajkr. Project management for implementing is into organizations. In T. Hruška, editor, *DATAKON 2005*, pages 173–197, Brno, Czech Republic, 2005. Masaryk University.

[29] Y. Wand and R. Y. Wang. Anchoring data quality dimensions in ontological foundations. *Commun. ACM*, 39(11):86–95, 1996.

[30] Workflow Management Coalition. Workflow specification, 2004. available at http://www.wfmc.org/standards/docs/Wf-XML-11.pdf.