System of Development Patterns in Service-Oriented Software

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 orientation is the leading paradigm of contemporary software. Each paradigm has specific practices and a specific set of design and development paradigms. For service orientation it holds too. We show that service orientation is a quite complex trend: There are several types of service-oriented architectures (SOA). The various SOA types may have different domain of application, different patterns and antipatterns, they can use different modeling and development techniques. Proper selection of SOA type can be a crucial task significantly influencing likelihood of project success. The applicability of individual SOA variants depends on requirements and on general business circumstances like staff knowledge, planned business alliances, and the need to reuse existing software. The proper selection of a SOA variant is an important pattern often made by the way. It is important that some patterns can depend on the effects of the other ones. Patterns should therefore be orchestrated. We discuss here mainly the patterns for the variant of SOA called confederation where communication partners need not be looked for. Most important patterns for confederations are user (business) oriented service interfaces, reuse of legacy systems and third-party products, and the use of so-called architecture services. Architecture services can serve as message transformers, heads of composite services, process managers, and integration constructs for the integration in the large. All architecture services discussed in this paper can be viewed as instances of one generalized concept from Petri nets.

keywords SOA types, SOA development patterns, user-oriented service interfaces, generalized Petri place, specification patterns, easy prototyping, interdependency of patterns.

1. Introduction

Service orientation is a paradigm having many aspects. It is manifested by the fact that the notions ”service orientation” and ”service-oriented architecture” are overloaded. Different people may assign different meanings to these notions. Although there are multitudinous meanings, we will focus only on a few – probably the ones being most important in practice. We shall discuss the variant of SOA being a virtual peer-to-peer network of peers behaving like real-world (human) services. It means that any service can offer capabilities as well as it can require them.

There are several variants of such SOA. They differ in the degree of autonomy of peers and their ”size”.

Various SOA variants have different structure of the collection of patterns they use. The selection of the patterns depends on the ”importance” of individual patterns. It can depend on the immediate technical effects of the patterns as well as on the business circumstances and plans like the use of third-party products, legacy systems, etc.

Some patterns can be blocked by business politics and business conditions like market alliances or the level of staff training. In this case it is good to know the possible losses caused by the rejection of a given pattern.

Some patterns are a precondition of the applicability of some other patterns. The most important case is the pattern building service interfaces so that they are ”usable” or user-oriented. It enables/implies the pattern ”coarse-grained interfaces” and prototyping via redirecting the destination of messages. Such a pattern is not generally known yet. The development of the collection of patterns cannot be any one-step process. This fact is often neglected.

Business needs and practical experiences led to an important change in the use of SOA-related communication protocol – SOAP (Simple Object Access Protocol, [34]). The shift is characterized by the increasing use of SOAP document-literal (SOAP-D) and decreasing use of SOAP-RPC (Remote Procedure Call) protocols [6]. In other words SOAP is now used as a XML-document carrying tool. Note

the documents can be semantically rich and well understood by users. It is, they can be usable. Usability is no matter of choice now. It has substantial consequences for the applicability of some powerful development techniques discussed below. Such effects are not generally known.

We discuss SOA patterns according their importance for the project success.

The structure of the paper is the following: A variant of SOA called Confederation is specified. It is shown that it is preferable to use in confederations the services of two types: application services having coarse-grained user-oriented interfaces and architecture (integration) services facilitating integration of other services. Several variants of the architecture services are presented. It is shown that all the variants are instances of one concept called Generalized Petri Place. It is shown that architecture services can be used to provide user-oriented interfaces or to support powerful prototyping. It is also possible to use architecture services to compose other services and processes. It is discussed how service orientation influences the specification patterns.

2. Choice of SOA Type

Crucial service oriented (SO) pattern is the choice of a proper variant of SOA. It should follow just after the decision whether SOA will be applied or not. We must – using the system environment – apply the variant of SOA best fulfilling the requirements. The type of SOA implies what further patterns are applicable, e.g., whether ESB (Enterprise Service Bus [5]) is good for the given system.

The concept of service orientation is in this paper pragmatically understood in the following way (see [23] for more exact definition[1]): A software system is service-oriented if it is a (virtual) peer-to-peer (p2p) network of loosely related (autonomous) components called services. The services somewhat behave like the services of real world. For example they are permanently ready to accept a request to do something. They can communicate with each other. Technically they communicate primarily by asynchronous message exchanges; synchronous communication can be an option. We can then say that such a system has a *service-oriented architecture* (SOA).

Service-oriented systems can have different architecture details depending on main goals of the systems and contexts in which the systems are used. Typical cases are:

1. e-commerce;
2. e-government, health-care systems, etc.;
3. small and middle-sized enterprises;
4. large enterprises, especially global decentralized organizations;
5. process control systems (soft real-time systems, some hard real-time systems);
6. systems logically having some features of SOA:
   (a) distributed applications being logical monoliths not allowing to apply full SOA. Common feature: communicating autonomous software components.
   (b) batch systems – autonomous software systems communicating offline or applying bulk data communication.

The autonomy of components is strongest in e-commerce systems and typically weakest in process control systems. In the cases 2, 3, and 4 the systems are formed by a core network of not too large number of services providing the basic capabilities of the systems (e.g., the services being wrapped information systems of individual offices of a state administration) and "peripheral" services providing e.g., portals on web.

SOA in large enterprises can consume large resources – money, people, and so on. There usually are powerful supervising authorities. The developed services forming the SOA are therefore in fact in large enterprises less autonomous than in small or medium firms. Large enterprises can moreover afford to develop the system from scratch or buy system like ESB and train people to use the new system.

Service-oriented systems integrate legacy systems, third-party products, and newly developed software artifacts. It is often required (see below) that the services have interfaces mirroring the languages of user knowledge domains. It is the main reason why SOAP-message literal is widely used.

In the case of the process control systems the system need not be open, the messages can have therefore formats based on remote procedure call (RPC) and middleware can be a proprietary one (e.g., based on system bus and primitives provided by operation systems).

2.1. Web Services

This version of service orientation endorsed by W3 Consortium focuses on web-oriented standards. These standards concern many aspects of service development and use.

The main idea is that the individual services should be strictly standardized to be able to communicate (or serve) to anyone (any other service) in the web world. The computerization should go so far that selection of cooperation partners can be done by services themselves.

[1] We, however, believe that this definition is too complicated and, may be, too restricting for applications of service orientation in some areas, e.g., in small or middle-sized enterprises, or in process control (i.e., real-time systems).

It is very promising. But there is also an opposite side of the solution: Implementation of all the standards (that are very complicated and moreover changing) is very complicated and can be reasonably done by quite large teams only.

There are several further issues with SOA based on web services. Web services must use universally applicable standards and such standards are difficult to be used properly by (human) users unless the use is based on libraries provided by large software vendors. But it leads to a dangerous situation called Vendor Lock-In Antipattern [4]. The standards like SOAP [34] and SOAP-related solutions tend to support the point-to-point communication rather than more complicated communication protocols and have a limited power to support orchestration of the services.

Web services seem to be the best solution for SOA systems like business-to-customer (B2C) *e*-commerce where communication partners must be looked for at the start of the cooperation. We call such systems *alliances* [18] for short.

The standardization of the semantics of communication messages is easier if the communication protocol is SOAP-RPC-based. It is a quite common practice that communication partners know each other permanently so the partners need not be looked for. In this case it is better to use a communication protocol based on exchange of documents as it enables the use of user-oriented interfaces discussed below. It is due to the fact that the documents can have syntax and semantics close to the language and knowledge domain of users. Such messages can be designed to be well understood by users. We say that they are *user oriented* or user friendly. It has many desirable consequences (compare e.g., [6]). An issue is that the semantics of the messages cannot be at present fully standardized and almost proprietary solutions must be invented. The above mentioned shift to document-based communication is an indication that confederations deserve substantial focus.

2.2. Software Confederations and Software Unions

Software confederation is a peer-to-peer network of loosely coupled services knowing each other. The communication between the services can be (and practice usually is) the high-level (declarative and coarse-grained) one. It makes sense to use semi-proprietary communication protocols like SOAP-D similar to inter-human communication specific for given problem domain. If the communication is SOAP-D-based, the documents should be in XML dialects.

Many existing service-oriented systems are confederations. Examples are information systems (ERP – Enterprise Resource Planning) of decentralized enterprises, advanced forms of CRM (Customer Relationship Management, [7]) and SCM (Supply Chain Management, [22]), e-government, health-care systems, etc. The principle to use user-oriented messages and user-oriented interfaces is probably the most important design and development pattern in certain SOA types. We call it *User-Oriented Interfaces* (UOI).

User-oriented interfaces have the following advantages. They are:

1. stable (rarely modified),
2. declarative (hiding implementation details),
3. good for agile development,
4. enabling easy integration of legacy systems and agile business processes
5. enabling easy implementation of screen prototypes,
6. improving system usability.

The main disadvantage of user-oriented interfaces is that they are usually not well standardized.

UOI is therefore a crucial SO pattern (a good practice, see [11] for definition). UOI covers functions like Facade[2] [11], but it substantially changes the properties of the system as a whole. Neglecting the use of UOI is itself an SO antipattern (a practice having usually undesired consequences; see [4] for definition). It is especially dangerous as people having object-oriented skills usually (according our long-time experience) do not see the prospects and opportunities of UOI. It is the SO antipattern "Well, What's New?" from [3].

The use of UOI is especially easy and desirable in confederations having a small number of highly autonomous core services. It is the case when legacy systems or third-party products are used. A good example is e-government, SOA-based information systems of small enterprises and municipal offices. We call such systems having such properties *unions*.

Unions are now frequently used by software vendors integrating large software artifacts being open source, legacy systems, third-party products, web services, and newly developed components.

3. User-Oriented Interfaces

The requirement that systems should have user-oriented (usable) interfaces implies that the interfaces of the services forming SOA must have specific property – they must be user-oriented (usable) as well. It is typical for the implementation of agile business processes (see below).

[2]Roughly speaking Facade is in object-oriented world an object providing common uniform interface to several other objects.

SOA, if used properly, is the first broadly used software development philosophy allowing a seamless integration of existing software system into new aggregates. It allows existing systems to be reused and it is not difficult to see that it is the only way to build information system of e-government [17] as well as the systems supporting global enterprises, health-care systems, and others. The resulting aggregates are typically confederations.

Let us discuss the case of e-government in details: The system of e-government must be as a rule constructed as an integration (interconnection) of the information systems of autonomous offices. The systems must be integrated with their local interfaces and without any substantial change of their already existing functions. The only feasible way of achieving it is to connect the systems to a middleware (in this case usually web – either Internet or a private network) enabling the communication between the systems. Business processes are in e-government called admministrative processes. Technically the admministrative processes do not differ from usual business processes.

The construction of the systems in such way has from managerial point of view substantial advantages – it allows saving of immense investments into existing (legacy) software. The proposed solution of the integration of the components (e.g., information systems) can be implemented almost unnoticeably by their existing local users. It saves investments into training and the expenses and loses caused by errors of end users during their adaptation to a new system. The resulting system tends to be a confederation, usually a union.

The service-oriented architecture is then a principle allowing the integration of software artifacts providing basic capabilities. They are often wrapped applications. We shall call them application services. The integration can be supported by architecture services discussed below. This attitude is a crucial SOA pattern.

Application services provide basic user domain capabilities. Application services integrated into a service-oriented system must be usually designed so that they can be easily used in business processes. The business processes must be agile – they should allow on-line users' involvement to be able to react to emergency situations. The users should be responsible for the business consequences of the processes. It is difficult to achieve if the interfaces are not user oriented.

User-oriented interface is as a rule coarse-grained and rather declarative, i.e., specifying rather what to do something than how to do it. It brings a pleasant benefit – the reduction of the load of the communication channels. Application services must and should be integrated as black boxes (like in e-government).

User-oriented interfaces of application services must be as a rule developed in close cooperation of developers and users. The system documentation can consist of the interfaces of application services only. It indicates that during the development of such systems many features of agile design and development of software systems can and should be used. At the same time SO enables the use of agile principles in the development of quite large systems [21]. It is typical for unions.

A very important advantage of user-oriented interfaces is that they mimic the interfaces of real-world services. The interfaces of real world services are often successfully used for a long time; some of them for decades and some even for centuries. Such real-world inspired service interfaces have a good chance, if formalized properly, not to be modified frequently. User-oriented interfaces enhance the usability [26] of the application services and also the usability of the entire system. Application services (e.g., legacy systems) are usually integrated together with their already existing local user interfaces. THe concept of usability of interfaces should be applied inside the system. It is not easy for IT experts to accept it as they must be able to be a bit skilled in user problem domain. Usability is advantageous for requirements specifications as well as for the agile software development processes.

4. Service Roles

Crucial property of services in SOA is that they all have technically the same properties. It is, they are all peers of a virtual peer-to-peer network. In confederations, however, using a logical view we need services providing functions supporting the integration of the application services. The services can provide the capabilities of Enterprise Service Bus or enhance them. They can provide broker services.

It can happen that the current interfaces of an application service are not user oriented, see the antipattern Chatty Services from [3]. Chatty services require many "tiny" messages per one meaningful action in the sense of users. The antipattern Chatty Services can be refactored (avoided) by the use of specific services called *front-end gates* (FEG). FEG is a service transforming user-oriented semantically rich messages produced by users or other services into series of fine-grained (implementation-oriented) messages required by the interface of a given application service and vice versa.

Front-end gate is one of the so called *architecture services* being the units facilitating the construction of service-oriented architectures. The existence of architecture services is an important feature of service-oriented philosophy. It can be viewed as a development as well as design pattern. Application services are often legacy systems. The pattern "Integration of Legacy Systems" is an important SOA pattern. It is, however, similar to the object-oriented antipattern Legacy Systems [4]. It is an indication that the

service-oriented philosophy is substantially different from the object-oriented one. FEG provides capabilities similar to Facade or Proxy patterns known from [11] but it has substantially different overall properties. FEG[3] is more an architecture pattern than a design pattern. For details see the section Generalized Petri Places below.

Application service is usually integrated as a black box whereas infrastructure services are usually newly developed and therefore integrated as white boxes.

To summarize the services in software confederations can of two basic types:

1. Application services (typically wrapped legacy systems or third-party products) providing basic capabilities (operations) of the system can be legacy systems, third-party products, or newly developed systems.

2. Architecture (or infrastructure) services supporting the integration of application services into a service-oriented system. (Note that the term "infrastructure service" has here the meaning different from the meaning used in ITIL methodology [14, 15], so we will not use it.) Besides the front-end gates we shall discuss the following architecture services: portals, data store services, process managers, screen prototypes, and generalized Petri places. All the services can be developed using very similar techniques and tools.

The acceptance of the concept of architecture services is crucial for the applicability of the patterns discussed below. The architecture services should be applied in all contexts except alliances and process control systems.

4.1. Architecture Services

Architecture services provide capabilities enabling various forms of integration of application services. The capabilities include enhancement of services interfaces and communication protocols, business process control services, or services acting as routers.

The use of architecture services is a very important development and design pattern of (confederative) service-oriented systems. It is possible that this principle can be successfully used in some alliances too.

Front-End Gates

If we want to reuse legacy systems or simply applications in SOA, the first issue to be solved is the reconstruction of their interfaces. The original interfaces are usually too fine-grained, disclosing implementation details, and often too

[3]FEG is a software service working as an interface adapter. It is therefore similar to the concept "service adapter" but its philosophy has specific features; see details below.

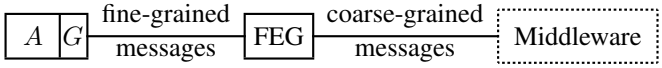


Figure 1. Connection of a front-end gate

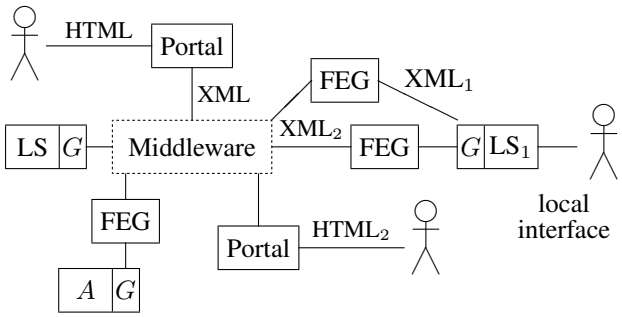


Figure 2. Multiple front-end gates

developer oriented. A very flexible solution is based on the technique of front-end gates (service adapters). The front-end gates provide capabilities provided in object-oriented world by object adapters (facade), proxies, and so on. The capabilities of front-end gates are, however, substantially more powerful than the ones offered by the object-oriented techniques.

The interface of an application A can be provided by none, one, or more front-end gates (FEG). A is accessible only through its FEG(s). FEG is a generalization of the concept of connectors in Enterprise Service Bus [30].

It is crucial that FEG is a peer of the virtual peer-to-peer network too. It is in fact an adapter service – compare object adapters in object-oriented world.

The resulting service-oriented system then can have the logical structure from Figure 2. Different FEG of a service can be used for different groups of its communication partners.

The development of FEG has a lot of common with the development of portals of the system as in both cases the task is to develop an automaton transforming k-tuples of input messages into m-tuples of output messages and sends them to (distinguished) destination services. The destination service consumes/processes them. We can use XSLT [33] or tools known for compiler construction for it. So we can conclude that the interface of (application) services can always be user-oriented, if necessary. The condition is that the software component providing services have properly designed "boundaries".

Data Store Services

Practical experience with SOA indicates that service-oriented systems must integrate batch systems and therefore

services requiring an implementation of data stores on communication channels. Some parts of such systems have then features of functional decomposition. Another application of data stores is the support of sophisticated communication protocols, sometimes more complex than the publish-subscribe one.

Data store services enable us to implement a part of service-oriented system in the way known from structured design – i.e., to apply main principles of functional decomposition. Note that functional decomposition is an object-oriented antipattern [4] but here it is an important pattern enabling a seamless integration of batch applications.

Examples when data store services are needed:

- Business process control service (see Process Manager service below) can use a data store to maintain and interpret business process control data.
- Some algorithms are too complex to be executed on-line. The components implementing such algorithms must then work in batch mode and their results must be stored in a data store possibly implemented as a specialized data-oriented service. A good example is scheduling algorithms in manufacturing systems [19]. The manufacturing scheduling algorithms are performed on enterprise level in batch mode as the algorithms are too complex to be started online. The schedules must be sent to workshop level in bulk mode and then possibly modified by a workshop manager or dispatcher.
- The communication must be supervised and possibly committed/blocked by users. It is typical for business processes if we need that they can be used in and agile way. Note that the agility is necessary if it is needed that the process owner is responsible for process consequences. Agile processes in small-to-medium enterprises must be agile to respond properly on business condition changes [20].
- Data store services can be used to implement complex communication schemas, e.g., the publish-subscribe one if not provided by the middleware yet. In this case the data memory component stores a set of messages. It can be used to solve the point-to-point antipattern [3, 16].
- There can be reasons to change dynamically the destination of a message due the facts known to users only (e.g., machine tool failures, incomplete data used by scheduling algorithm, etc.). The changes can be performed by a user or by an application/service or by activities of destination services.
- Data store services can be used to implement functions necessary for debugging.

Data stores can therefore be used to enhance middleware functions, they can be used to integrate batch applications and can serve as a powerful enhancement of business processes control, especially in the cases when a given function can be provided by several application services that need not provide the same collection of elementary services or provide the same elementary services but with different quality (see [19] for details).

Process Managers

Business processes must often require on-line involvement and supervision of process owners into their operation. This feature is known as agility. Agility is a very desirable feature of business processes – especially in small and middle-sized enterprises. It is the condition for the requirement that process owner should be responsible for business losses caused by process steps as well as for on-line process changes performed by the owner. The reasons are (compare [20]):

1. The process model/definition is based on data that need not be for various reasons timely, accurate, or complete. The business conditions may also change.
2. The process owner can be obliged to commit some risky process steps.
3. The information on the process should be understandable for experts (not necessarily IT ones), e.g., at a court judging a business case.
4. The process model M should be stored as a part of business intelligence and updated by users.
5. It is desirable to be possible to have process model in different languages, if necessary, e.g., in BPEL [2], Aris, [13], workflow [36], or in a semistructured text.

As it is not desirable to have centralized services in peer-to-peer systems (compare experience with UDDI [32] and with UDDI-based world-wide repositories) we can use the following solution (see Figure 3):

1. During the process enactment a new service instance called *Process Manager* (PM) is generated on the request of the process owner O. During the generation of PM a process model M (if any) is transformed into a process control data C using parameters provided by O. O can possibly generate C directly without M. M can be copied from a data store. A solution when M contains no data or when O is a proper textual document used by a process owner, is also possible. C can be stored inside PM.

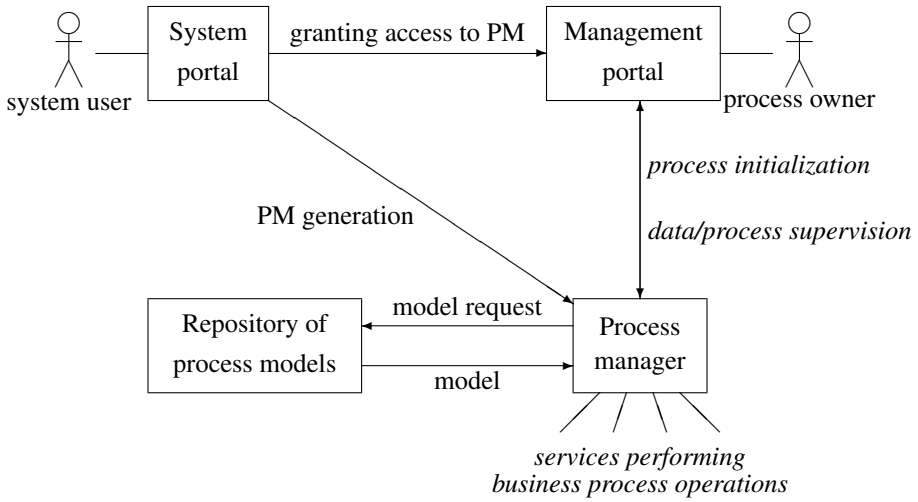


Figure 3. The use of Process Manager

2. During the operation of the process PM generates (using C) service calls. The calls can be synchronous (call and wait for answer) or asynchronous (just send a message). C can be modified on-line by process owner, if necessary.

3. It is important for the reasons discussed above 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 – it is, the interfaces are user oriented. We have seen that user-oriented interfaces have many software engineering advantages. Note that Process Managers are from logical point of view portals of subnetworks of services providing the operations of the business process. The subnetwork behaves like a portal SOA.

Portals

It is advantageous to design a portal (system user interface) as a service (peer of the network) providing the user interface to the some functions of the system to a specific group of users. Any system can have several portals.

In the case of process managers it is meaningful to generate not only the manager but also a portal (user-interface) for it. Another implementation can be via portlets plugged into a portal. According to our experience it is a less flexible solution.

Portal SOA A service in SOA can in principle communicate with any other service. This possibility can be reduced according to service role it plays in the system according the principles of the design and implementation of a given variant of SOA.

The simplest version of SOA has services of only two types: application services and (usually) one portal. The application services can communicate with and using the portal only (Figure 4). Such solution is called *portal SOA*. Note that the application services can be structured, they can be again (virtual) networks of subservices.

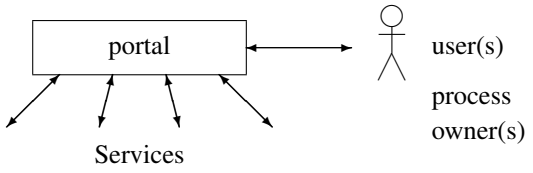


Figure 4. Portal SOA: logical view

A SOA system can have subnetworks possessing different SOA construction principles. An example is inclusion of multiple e-commerce (sub)systems in a large ERP. Another example is the subnet providing support of a business process. The head of the subnet can be a proper service – for example a process manager service PM discussed above. Often the services can be composite services.

Portal SOA is recommendable in the situation when the application services are very autonomous and the agility of business processes is desirable, the application services can be equipped by user-oriented interfaces and the system response times need not be too short. Such conditions are fulfilled quite often. The implementation of portal SOA is simpler than the development of general SOA. It is therefore important to detect whether the system to be developed can have the architecture of portal SOA.

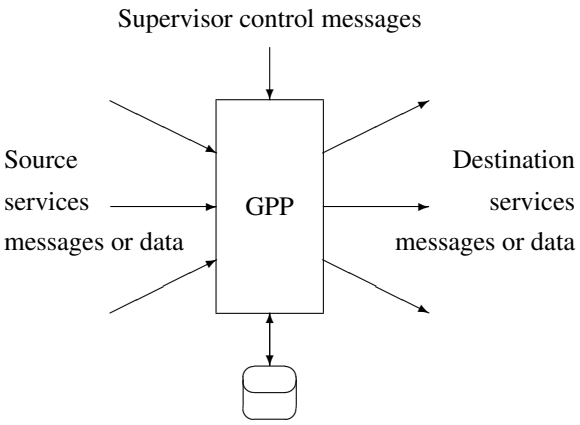


Figure 5. Generalized Petri place (simplified)

Generalized Petri Places

All the above discussed architecture services (with a partial exception of portals) can be viewed as specific variants of the service type called *Generalized Petri Place* (GPP). GPP transforms tuples of input messages into tuples of output messages. It can have its local data store (Figure 5). The functions of GPP, e.g., message routing, can be influenced by a (human) supervisor. GPP is a generalization of the concept "place" in colored Petri nets [27]. It is possible to use tools like an XSLT [33] engine to generalize the implementation of a front-end gate to have m "inputs" and n "outputs" (Figure 5). In other words: a front-end gate can be transformed into a generalized transducer transforming m-tuples of input messages from several sources into n-tuples of messages sent to several destinations.

The functions of GPP are similar to the ones of Facade [11] but GPP is a more general concept as the syntax of messages can be substantially changed by GPP and the messages themselves can be declarative and therefore not procedure-call oriented. The links from Figure 5 can be dynamically changeable at runtime. GPP can easily implement the functions of many other patterns from [11] like Build or Abstract Factory. The functions and behavior patterns are easily changeable at run-time as almost no source code changes and recompilations or no relinking is necessary to change the behavior. Redirecting of messages is needed e.g., for screen prototypes discussed below or in emergency situations. Redirecting can be set up on a request of a human supervisor.

On the other hand the power of GPP can be a dangerous tool in hands that are not skilled enough as there is a little syntax overhead. It is, however, well known that coding (programming) is no bottleneck of software development. The bottleneck is the requirements specifications [31]. The root reason of the problems is the snags in cooperation with users and it is almost not needed here.

A specific variant of it is the use of GPP as an entry point of a subnetwork of services (to assemble into a composite service). GPP then plays the role of a FEG of this composite service. It suffices to require that any message sent to any service of the subnetwork must pass the GPP being FEG of the composite service. This technique is then a service composition tool. We call such a GPP the Head of Composite Service.

GPP can be used to integrate several service-oriented systems. GPP then can serve as a hub enabling the integration. GPP can also connect subnets having properties of alliances, e.g., if the peers of the subnet are web services.

The systems using GPP can therefore have a very rich structure.

4.2. Operations on Generalized Petri Places

We have noted that all above discussed variants of architecture services can be viewed as modifications or special cases of generalized Petri places. Let us now systematize the types of modifications. Variants of communication protocol:

Standard (or public) protocols. The protocols used as a basic variant of communication in given SOA.

Bulk communication. This is a variant of communication used for the communication with batch systems.

The role of data store. We have discussed the following cases:

- massive data store filled in bulk mode,
- data store of messages,
- log memory,
- data store (repository) of the models of business processes.

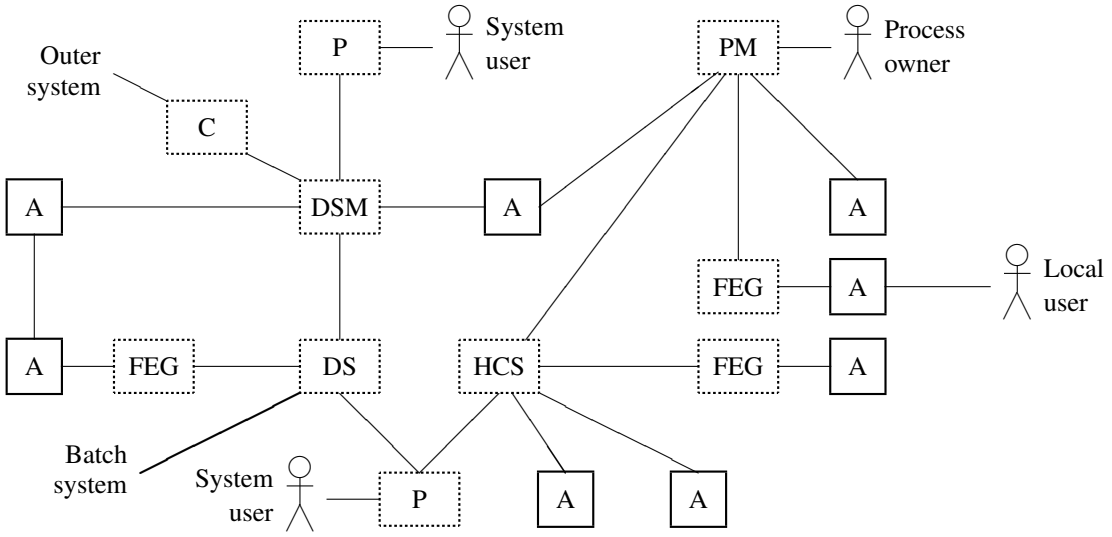
Message paths. We discussed the case when messages must go through a Head of Composite Service and the case of Connector. In fact in Portal SOA the application services communicate via Portal only.

The roles of human interfaces:

- observing and administration only,
- intelligent human interfaces.

The lifetime of services – generable and destroyable vs. permanent ones.

The transformations can be combined, so we can have a broad set of architecture services. It is a topic of further research to find out whether it can lead to further variants of architecture services.



DS – data store, DSM – data store of messages, FEG – service adapter, A – application, P – portal, PM – process manager, C - connector, HCS – head of composite service

Figure 6. Structure of SOA with architecture services

4.3. Architecture Services in Action

The architecture services discussed above enable us to design flexible service-oriented system having flexible and open logical structure that can be easily changed. Example of such a service-oriented system is in Figure 6.

The virtual p2p network in Figure 6 has two subnetworks. One comprises a service providing business operations for the architecture service Process Manager (PM). The second is a composite service headed by a Head of Composite Service (HCS).

Note, however, that the services providing operations for the business process controlled by PM can provide capabilities for other processes if appropriate.

Figure 6 shows how flexible and powerful is the service-oriented paradigm. It can be, however, dangerous if not used with caution.

A very important SOA pattern is a proper use of universal middleware components and the proper use of architecture services. ESB is not broadly used in unions. The reasons are not clear yet. Note, however, that ESB can imply Vendor Lock-In antipattern.

4.4. Fuzzy Tiers in Confederations

If we summarize the above discussion, we can distinguish the following tiers in a confederation: basic message-transport middleware, partly programmable middleware (typically Enterprise Service Bus – ESB), middleware enhancements (FEG, some Data store types, etc.), application services, composite services, orchestration, and system portal(s). Orthogonal to these tiers are local user interfaces. So the system has the structure from Figures 7 and 8. Problem is that there are cases where the services provide functions for more than one layer (tier) – for example data stores. These services can serve as a middleware enhancement as well as a process manager. It is a service orchestration tool. GPP can serve as a service orchestration tool as well as a service composition tool.

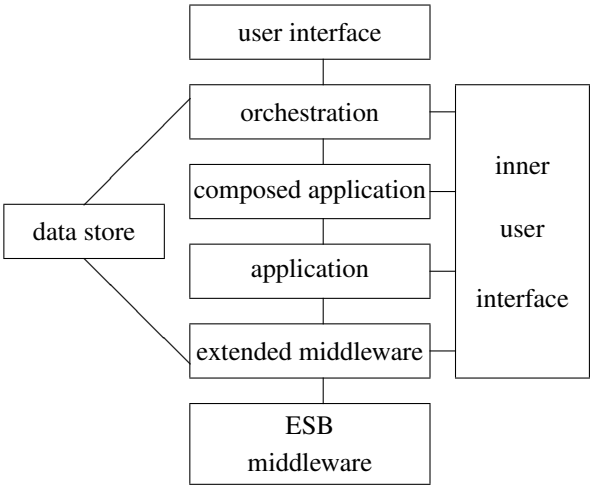


Figure 7. Tier view of a confederation

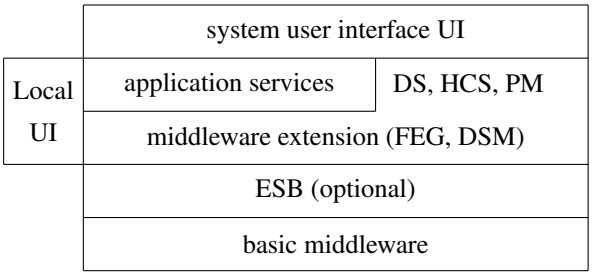


Figure 8. Tiers in SOA

4.5. Screen and Simulation Prototypes

Native service-oriented development process is the incremental one. An issue is how to simulate a service S not implemented yet. As it suffices to simulate the communication with S, we use the following procedure:

1. The middleware or a GPP can change the destination of (redirect) some messages. The messages being preferably in XML format to be sent to S are redirected to user interface UI (portal).

2. UI responses like the S would have responded.

This solution can be generalized to test response times in service-oriented process control system (real-time control systems). In this case the messages are sent to a Simulator. The Simulator can be either a hardware device or a simulator program written in a simulation language or in a language similar to C. The simulation program uses a Calendar of Coming Events (CCE) known from discrete event simulation languages. CCE can be programmed in e.g., C++ or even a discrete simulation language can be used. Details can be found e.g., in [19].

The implementation of the screen prototype is substantially simplified if the interfaces of the application services are user-oriented and uses XML documents (see SOAP-D above).

4.6. Crucial Vision and Specification Pattern

We understand SOA as a virtual peer-to-peer network of software components behaving in some sense like real-world services. The peers are called (software) services. If there is no danger of misunderstanding, the term *service* will mean software service. This broad definition covers quite different systems, the main goals/visions of which can be quite different. Different visions then imply different marketing and technical (engineering) properties, the system of patterns inclusive.

Every SOA system[4] (SOA in the above sense) consists of (compare [20]):

- application services providing basic "atomic" business capabilities, atomic means a software system providing "basic" business capabilities viewed as a black box. An example of an application system is an (encapsulated) legacy system. Application services are usually integrated as black boxes. The development from scratch can be used if necessary or appropriate.

- middleware providing tools for transport of messages between (i.e., supporting the communication of) the peers

- architecture services enhancing the capabilities of middleware. Examples are service adapters and portals. Architecture services are usually developed from scratch, i.e., they are white boxes.

The properties of all the three tiers substantially depend on the global system goals. It then implies what patterns are applicable. Let us give some examples.

1. e-commerce systems. The aim is to support world-wide business activities. It follows that a world-wide network must be used to implement the middleware. It is feasible only if no proprietary standards are used. The use of architecture services is very limited. Systems are very open. The use of web services is appropriate.

2. Process control systems. The main aim is very secure software developed almost entirely from scratch. The number of services is limited. The details of communication protocols can be agreed, the architecture services are usually not used. An exception is the use of portals (client/user tier). The service interfaces can be fine-grained and IT developer oriented (e.g., using R-PC philosophy).

3. System supporting large (partly) decentralized organizations (e.g., e-government, municipal authorities, health institutions and networks, etc.) and small to medium enterprises. The main aim is the reduction of the maintenance of such systems and integration of autonomous. The system consists of large application software services. It is preferable to use architecture services enabling a seamless integration of the services supporting organization units or the integration of third-party products. Some parts of such systems can support e-commerce. These subsystems then have the properties specified in the point 1. Such systems are quite frequent and are in fact the engine of global economy.

[4] SOA system is an abbreviation for "system having SOA".

4. Portal SOA. The communication of services must be controlled or supervised by users. In this case it can be good to generate service requests by portal The capabilities of middleware are used not too much. Service adapters (front-end gates, FEG) can be useful. Such an arrangement can be used for the implementation of agile business processes (see [28] and below). The crucial pattern is to start with the decision what variant of SOA is to be used and what application services and architecture services should be used. For the reasons discussed below the software artifacts implementing application services are as a rule large and the service interfaces are coarse-grained.

4.7. Further Notes on the Choice of a Proper SOA Type

The crucial decision is the proper selection of optimal SOA type. We often have no choice. We must use different solutions for large critical systems than for an information systems supporting a small enterprise.

On the other hand the properties of SOA systems supporting business in a small enterprise and in a very large one must be surprisingly similar although due different reasons:

SME have limited resources, so it must (re)use legacy systems as much as possible. The functions of services must be user-oriented to be used properly as there are few, if any, available IT experts able to understand user needs. User-oriented interfaces are necessary for agile business processes. They are welcome if the responsibility of business process owners for their processes is required.

Large enterprises have more resources but large changes can be too time consuming and implying too high burden on end users. Agility of processes is desirable and responsibility are needed. User-oriented interfaces support information hiding in the sense of software engineering. It is good for in- and out-sourcing.

We conclude that the proper detection of the SOA type is crucial architectural and requirement pattern. It is crucial in the sense that not applying it implies fatal antipattern. Note that in business the unions are often the only possibility.

5. Conclusion

The most important property of SOA is that it is a virtual peer-to-peer architecture. It is a quite broad definition, broader than all the SOA variants defined in standards by e.g., OASIS and W3 Consortium. It includes also quite dynamic structures not having all the properties required by the standrads. For example, it allows to integrate batch systems, web services (either complying or not to the W3C web services standards), and other systems not satisfying the requirements of the standards. Too strict definitions of SOA and software services could be the reasons of partial dissatisfication with SOA observed in the last year.

We have shown that we can adapt useful SOA solution not fulfilling the requirements of complex definition typical for many SOA-oriented standards. Such solutions can successfully support the development of information systems supporting ERP of small or middle-sized enterprises. We can even apply solutions not leading to pure peer-to-peer networks. We can use tools like MQ by IBM or solutions offered by operating systems like named pipes. It is a big promise and challenge that is often missed.

We require only that all the services (peers) of a SOA have structurally similar properties. The overall logical structure of SOA is an implicit consequence of the inner functionality of the particular services and their communication rules in the way discussed in this paper. We believe that the importance of this almost obvious property is significantly underestimated and often not taken into account at all.

It is still open what further architecture services should be invented and used.

SOA could be useful in business only if the user involvement is taken into account during development and allowed during use of the system. It implies the use of user-oriented interfaces of services and application of certain features of agile development. User involvement and ROI imply that the main SOA development and even specification patterns in business should be the integration of legacy systems.

If we look into the history, we see that the problem of year 2000 (Y2K – problems with changes in immense number of COBOL programs necessary due to century change) was the consequence of the use of legacy systems written in COBOL and used for decades with almost no maintenance. It caused the problem that there were no COBOL programmers able to make the changes. It is desirable to develop systems needing almost no maintenance. The use of legacy systems is the way to the true reusability (substantial than in the object-oriented environment – compare [10]). It is difficult to assume that the modern software should not use legacy services requiring almost no maintenance. SOA enables it.

The vendors must, however, adapt their marketing strategies to the challenges of SOA revolution. Users must develop new skills and develop new business processes able to benefit from the power of the service-oriented paradigm.

The main contributions of the paper are the following:

1. It is shown that it is good to study SOA variants used especially in small firms and called confederations and unions.

2. The criteria for the selection of the variants are specified.

3. The detection of the dependencies among patterns used for the development of confederations and unions.

4. Analysis of the importance of user-oriented service interfaces and proposal how to implement them.

5. Powerful system development prototyping.

6. Concept of services and design of the most important ones.

7. The development of the concept of generalized Petri place and treatment of the architecture service as instances of the generalized Petri places.

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. Crucial patterns in service-oriented architecture. In *Proceedings of ICDT 2007 Conference*, page 24, Los Alamitos, CA, USA, 2007. IEEE CS Press.

[2] 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.

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

[4] 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.

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

[6] F. Cohen. Discover SOAP encoding's impact on web service performance. *developerWorks*, Mar. 2003. http://www-106.ibm.com/developerworks/library/ws-soapenc/ 2009-05-15.

[7] J. Dyché. *The CRM Handbook: A Business Guide to Customer Relationship Management*. Addison Wesley Professional, Boston, 2002.

[8] T. Erl. *Service-Oriented Architecture – A field Guide to Integrating XML and Web Services*. Prentice Hall, 2004.

[9] T. Erl. *SOA principles of Service Design*. Prentice Hall, 2008.

[10] L. Finch. So much OO, so little reuse. *Dr. Dobb's Journal*, May 1998.

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

[12] Gartner Inc. Gartner says the number of organizations planning to adopt SOA for the first time is falling dramatically, Nov. 2008. http://www.gartner.com/it/page.jsp?id=790717 2009-05-15.

[13] IDS Scheer. Aris process platform.

[14] International Standards Organization. ISO/IEC 20000-1:2005: Information technology – service management – part 1: Specification, 2005.

[15] International Standards Organization. ISO/IEC 20000-2:2005: Information technology – service management – part 2: Code of practice, 2005.

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

[17] J. Král and M. Žemlička. Electronic government and software confederations. In A. M. Tjoa and R. R. Wagner, editors, *Twelfth International Workshop on Database and Experts System Application*, pages 125–130, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[18] 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.

[19] 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.

[20] 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.

[21] J. Král, M. Žemlička, and M. Kopecký. Software confederations – an architecture for agile development in the large. In P. Dini, editor, *International Conference on Software Engineering Advances (ICSEA'06)*, page 39, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[22] B. Lowson, R. King, and A. Hunter. *Quick Response: Managing the Supply Chain to Meet Consumer Demand*. John Wiley & Sons, New York, 1999.

[23] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz. Reference model for service-oriented architecture 1.0, committee specification 1, 19 July 2006, 2006. http://www.oasis-open.org/committees/download.php/19361/soa-rm-cs.pdf 2009-05-15.

[24] E. A. Marks and M. Bell. *Service-Oriented Architecture – A Planning and Implementation Guide for Business and Technology*. John Wiley & Sons, Hoboken, New Jersey, USA, 2006.

[25] J. McKendrick. Gartner: SOA sinking into trough of disillusionment, Nov. 2008. http://blogs.zdnet.com/service-oriented/?p=1211 2009-05-15.

[26] J. Nielsen. *Usability Engineering*. Academic Press, New York, 1993.

[27] C. A. Petri. Kommunikationen mit automaten (Communication with automata, in German). *Schriften der IIM*, (2), 1962.

[28] A. Schatten and J. Schiefer. Agile business process management with sense and respond. In S. C. Cheung, Y. Li, K.-M. Chao, M. Younas, and J.-Y. Chung, editors, *ICEBE*, pages 319–322. IEEE Computer Society, 2007.

[29] D. Sholler. 2008 SOA user survey: Adoption trends and characteristics, Sept. 2008. http://www.gartner.com/DisplayDocument?id=765720 2009-05-15.

[30] Sonic Software. Enterprise service bus, 2004. http://www.sonicsoftware.com/products/sonic_esb/ 2009-05-15.

[31] Standish Group. Chaos: A recipe for success, 1999.

[32] UDDI Initiative. Universal definition, discovery, and integration, version 3, 2002–2003. An industrial initiative, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3 2009-05-15.

[33] W3 Consortium. XSL transformations (XSLT), 2007. W3C Recommendation. http://www.w3c.org/TR/xslt20 2009-05-15.

[34] W3 Consortium. Simple object access protocol, 2000. A proposal of W3C consortium. http://www.w3.org/TR/SOAP 2009-05-15.

[35] W3 Consortium. Web services activity, 2002. http://www.w3.org/2002/ws/ 2009-05-15.

[36] Workflow Management Coalition. Workflow specification, 2004.