Flexible Business-oriented Service Interfaces in Information Systems

Michal Žemlička[1,2] and Jaroslav Král[2,3]
[1]*Department of Software Engineering, Faculty of Information Technology, Czech Technical University in Prague, Praha, Czech Republic*
[2]*Network and Labs Management Center, Faculty of Mathematics and Physics, Charles University, Praha, Czech Republic*
[3]*Department of Computer Systems and Communications, Faculty of Informatics, Masaryk University, Brno, Czech Republic*

Keywords: Flexible User-driven Integration, Architectural Services, Large Software System in SME, User-oriented Service interfaces, Flexible SOA.

Abstract: Information systems supporting flexible business in small-to-medium enterprises must be easily modifiable under the supervision of their users. The users (business people) must take active part in agile system development and maintenance. The systems must be able to integrate large legacy systems and to communicate with the systems of independent business partners. Business processes need not be executed by a single ERP. We discuss a variant of SOA able to meet these requirements. The discussed SOA uses communication protocols based on problem-oriented languages. We propose a concept of organizational (architectural) services generalizing the concept of connectors and routers. The power and usefulness of the proposal is demonstrated on the examples of service composition, business-oriented interfaces, agile business processes, portals, and gateways. The proposal is based on experience from practical SOA projects.

1 INTRODUCTION

Information systems are permanently becoming more complex. They must support agile business processes, must have a proper architecture, and cannot be developed and maintained using classic software engineering attitudes (Royce, 1970; Yourdon, 1988; Sommerville, 2010). The crucial reasons are:

- Large information systems cannot be developed and maintained within reasonable costs and deadlines if a single-step process is used. The system must enable agile changes of business processes and agile maintenance.
- System users must often take part in the system supervision, development, and maintenance. They must therefore understand the communication among system components.
- It is necessary to integrate large legacy systems and third-party products to reduce expenses, meet terms, and to preserve business knowledge.

We will show that the requirements can be met if the developed systems have the following key features:

1. The systems have a service-oriented architecture with coarse-grained components having coarse-grained business-oriented interfaces.
2. The services are connected by a virtual communication network of active elements being again services. We call them *architectural services* and the systems using them *confederations* (Král and Žemlička, 2013).

The concept reflects the needs of small-to-medium business. We will discuss the structure, features, and implementation principles of some architectural service types that have proven their quality in practice. We will summarize the reasons why confederation is a very powerful generally applicable concept.

Structure of the paper: Section 2 gives an overview of the critical issues faced by current business information systems. Section 3 deals with architecture patterns solving the issues. The last section summarizes the results and gives recommendations.

2 BUSINESS-ORIENTATION AND INTEGRATION

A proper architecture enables almost independent development of system components. It is not the only

Žemlička M. and Král J..
Flexible Business-oriented Service Interfaces in Information Systems.
DOI: 10.5220/0004888901640171
In *Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2014), pages 164-171
ISBN: 978-989-758-030-7
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

advantage. Below proposed architectures:

- allow easy integration and replacement of legacy systems and third party products,
- simplify insourcing and outsourcing,
- cheapen, fasten, and uprate the development,
- enable agile business processes and agile development,
- make the system development and maintenance more efficient; users could take part in system development, maintenance, and operation; some task can be solved by users alone.

2.1 Small-to-medium Business

Small or medium-sized enterprises and organizations (SME) form a significant part of contemporary economy. SME must do their business processes (BP) in a globalized market and global social environment. BP of SME must promptly react on the dynamic changes in business environment.

The dynamics of the business processes in SME requires almost always a direct active participation of business people and managers. They must have the possibility to utilize their business knowledge, experience, abilities, and skills. Smaller companies usually do not have enough IT people having necessary business knowledge. There can be budget issues. Last but not least, the business culture in SME is different from the culture in big companies (Robbes et al., 2013).

It follows that the modification of the processes must be cheap and transparent for business people. It also means that legacy systems and open software must be often used.

Business processes in SME are usually open. It is, they often cross company borders. Some parts of the processes must then be done by independent organizations using classic business procedures based on the rules and documents of commercial cooperation. Business partners can be changed dynamically. It is then desirable to allow direct cooperation between information systems of the business partners.

The information systems of cooperating business partners behave like (composite) SOA services using a document-oriented communication. We point out that the documents are for us coarse-grained pieces of communication well understandable by users. They can eventually have the form of documents in the sense of electronic data interchange (EDI) if appropriate. XML-based formats are often preferred now.

The SOA uses organization services acting as wrappers, routers, gateways, orchestrators, and user interface. The message paths can be influenced by business process owners. The individual ERP can use for the steps of the processes the same approach.

2.2 Business-oriented Systems

Human aspects are crucial for the formulation of the aims and requirement specifications of information systems as well as their development and use. Information systems influence real world processes. These aspects are especially important in the case of information systems supporting business processes.

The dynamic character of modern economy implies that the business processes ought to be agile, i.e., easily modifiable by users. The business partners can be changed dynamically. It, together with the growing complexity of software, implies that the system must be able to integrate/use existing large legacy applications and to utilize already existing informal personal knowledge and social business contacts. It avoids risky business process restructuring.

Taken together the communication should have the form of an exchange of legible messages and the system as a whole has a specific service-oriented architecture (confederation). It uses coarse-grained business-oriented communication between coarse-grained software components.

The documents used for service to service communication inside an information system and between information systems should be intuitively understood by users. It can be digital variants of the (business) documents. Their form can be agreed by communicating parties. The standards like EDI can be used but there are reasons why EDI has not been generally accepted. The transparency of communication and the agile choice of business partners are otherwise hard to achieve. It allows agile on-line modifications of business processes. It is necessary to support agile choice of message addressees – of business partners.

We will discuss technologies enabling encapsulation of software artifacts using proxy services solving the above mentioned requirements. The technologies further enable easy implementation of generalized capabilities of Business Process Model and Notation (BPMN, (Object Management Group, 2011)) – understandability, openness, gateways, etc.

The proposed solutions are good to support business processes in back office as they use business-oriented interfaces. It solves issues from (Boyd et al., 2012a; Boyd et al., 2012b).

3 SOFTWARE CONFEDERATIONS

Software confederations are service oriented systems with loosely coupled coarse-grained services being in principle of three types:

1. **Business Application Services:** encapsulated business applications and business systems supporting basic software operations (example: accounting service);
2. **Architectural Services:** allowing creation and changes of the architecture of the built system (example: proxy/generalized application wrapper implemented as a service – front-end gate; see below);
3. **SLA Services:** organizational business services at the SLA level (e.g., identity management implemented as a service) where service requester must wait for the response of service provider, i.e. the services communicate synchronously.

3.1 Basic Service Types

We will describe service types most frequently used in confederative systems. The descriptions take into account both development and maintenance point of view.

3.1.1 Business Application Service

Confederations have a two-tier structure. The basic tier is the collection of services providing basic business capabilities. A typical variant of such services is a properly wrapped business application or even entire information systems of business partners. The wrapped entities usually have a legacy interface supporting a direct communication with users and with other applications (Fig. 1).

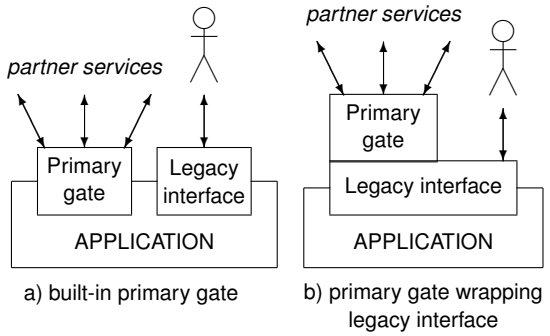

Figure 1: (Business) application service with primary gate and legacy interface.

The services must be by definition capable to take part in peer-to-peer communication. It is preferable to implement this capability using a component called *primary gate* (Fig. 1). Primary gate can eventually utilize (a part of) the legacy interface (Fig. 1b).

The communication with partner services provided by the primary gate is as a rule due many reasons fine grained and remote procedure call oriented. We have, however, seen that the interfaces should be (like their real-world counterparts) document and user knowledge domain oriented. We therefore need a tool able to transform these different communication protocols. It can be achieved using a specific organizational service type called *front-end gates* described below, see also Fig. 2. These services provide the capability of adapters (compare (Newcomer and Lomow, 2005; Krafzig et al., 2004; Reynolds and Antony, 2010). Their construction as services significantly increases system variability and flexibility.

Application services can communicate using intelligent coarse-grained communication protocols. The communication is provided by network of organizational (architectural) services forming the second tier of the system.

The use (integration) of existing applications simplifies and shortens the development and reduces the risk that the newly developed system will not do what is required. On the other hand there is a risk that the application can be changed by a third party or that it cannot be used for other reasons.

In such a case it is possible to replace it by another properly encapsulated application. The replacing application can be newly developed. If business-oriented service interfaces are used, it is likely that change propagation can stop at the front-end gates. If the interface changes are not requested or needed by the partners, the business-oriented service interface available for them usually need not change.

This service interface stability significantly improves stability of the system and simplifies and decomposes system maintenance.

Throughout this paper the communication is mentioned generally (typically without restriction of type or means). If necessary, the restrictions are included into the figures.

3.1.2 Basic Architectural Services

Architectural services are organizational services that can be viewed as a generalization of connectors in the sense of (Mikic-Rakic and Medvidovic, 2002). Long term experience with the use of architectural services in manufacturing control systems (Král et al., 1987; Král et al., 1979) and elsewhere, see, e.g., systems

produced by ICOS Praha and logistics systems produced by ICZ, proved the usefulness of the concept. We currently study applicability of our approach in e-government using virtual filling rooms.

The transition from classic connectors to architectural services looks conceptually simple: it is just an implementation of the connector as a service (CaaS). The issue is that the communication with the connector as a service must be sometimes asynchronous and must use multipoint protocols. It may imply limitations for the use of synchronous communication protocols. Our experience from practical and educational projects shows that mastering this paradigm is often very tough. Some people doubt that such an approach can work – although the cases of its successful application (industrial projects inclusive) were described many years ago, see e.g. (Král and Žemlička, 2004; Král et al., 1979). In comparison to classic connectors CaaS have the following advantages:

1. higher independence of protocols enabling various use policies;
2. better opportunities to interconnect almost independent components;
3. possibility of agile supervision of system operation by users and managers;
4. new incremental development methods and especially agile variants of maintenance.

We present here architectural service types we have already used or that support especially interesting capabilities. Connectors as services play in confederations the role of the Petri net "places" used in parallel automata models (Petri, 1962; Jensen, 1997). It could be an issue for the developers used to use object-oriented paradigm. There are many situations (like broadcasting or message composition) where the asynchronous communication is necessary.

Front-end Gate

Front-end gates are special services translating interface from the form requested for the integration (matching partner needs) to the form supported by the application service primary gate and vice versa. It is possible to use techniques originally developed for transducers or compilers ((Aho and Ullman, 1973; Grune and Jacobs, 2008; Žemlička, 2006)). XSLT (W3 Consortium, 1999) can be applied in the case of quite simple transformations of XML-based message formats. Front-end gates can be equipped by capabilities enabling business process owners to redirect the communication and to log it (Fig. 2). For the sake of simplicity these capabilities are not shown in most figures.

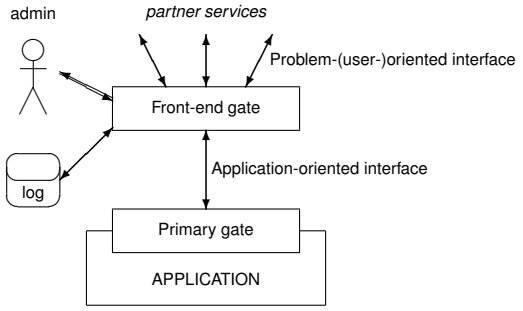

Figure 2: Application service and its front-end gate.

Front-end gate in Fig. 2 wraps the application service to hide peculiarities of application service interface. It is a very powerful implementation of Parnas's recommendation on information hiding (Parnas, 1979).

An application service can be easily replaced by another one. It greatly simplifies the integration of legacy systems, third-party products, and open source artifacts and also almost independent development of the services.

The transparency of log records is preferable for the business management and for solving business issues and enhancement of business intelligence. So the proposal enables the implementation of all the capabilities mentioned in Section 2. The capabilities are generally applicable. The pattern from Fig. 2 offers further interesting generalizations, see e.g. Fig. 3.

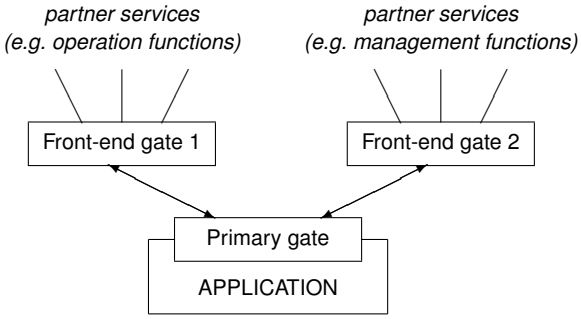

Figure 3: Application service and with multiple front-end gates.

The communication between partners and a front-end gate can be supported by other means than the communication between the front-end gate and the supported service if necessary. The front-end gates can serve also as service connectors for various communication means or as middleware gateways. The basic principles of the concept of FEG was introduced in (Král and Žemlička, 2002). The wrapper in (Erl, 2009) has similar but less general abilities.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Portal

For a significant group of users the access point to the system is provided by a special service called *portal*. It helps the users to specify what they need and how to tell it to the system. It also presents the users the reactions of the system.

In more detail: The portal usually presents the user an overview of available activities. If the user selects one of them, the portal should lead him/her through the specification of necessary and optional parameters. The portal then gives the collected request to the services that are assigned and able to process such requests. If the request is incomplete or imprecise, the portal should lead the user through the additional requirements. In some cases it is reasonable to pass the non-standard request to individual processing (emergency, rare cases).

From the more technical point of view a portal is a service providing usually web-based user interface allowing users to check all their service requests and responses. From the system side it is a service sending and receiving messages with various access rights.

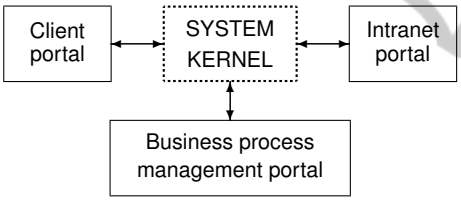

Figure 4: Example of a system with multiple portals.

It is often reasonable to equip confederative information system with multiple portals. Typically there will be specific portals for employees (intranet) and for customers (extranet) and there can be various special portals for special user groups (e.g. process owners).

It is also possible to encapsulate various portals into composed ones. The composed ones can delegate handling of particular user interfaces to particular portal services and handle only integration of the user interface into a single access point (Fig. 5). Data and computation intensive interfaces should be provided by specialized portals. The separation of concerns reduces demand on their throughput and simplifies their development, maintenance, and use.

From the maintenance point of view the separation of complicated user interface agendas into separate services significantly simplifies the localization of errors and their correction.

The solution is logically similar to portlets (OASIS, 2008). The encapsulation of subportals into special services allows a better load balancing and a higher overall throughput.

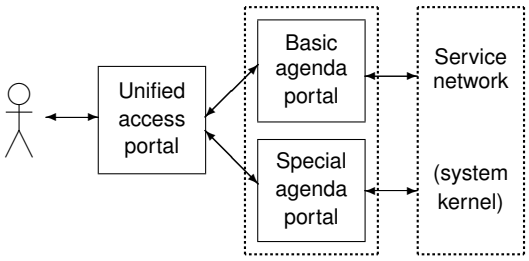

Figure 5: Grouped multiple portals.

3.1.3 Advanced Architectural Services

Head of Composite Service

Larger systems tend to an integration of logically closed smaller parts. Such integrations are usually layered or hierarchical. Real-world services tend to be integrated hierarchically, for systems supporting them we decided to use also hierarchical integration.

We call logically closed groups of services that can behave as logically single service *composite services*. The group is equipped by a special service representing the group and providing communication of the group with "the rest of the world". All the communication of any of the services in the group to and from any service outside the group must go through this service. This policy can be weakened (modified) easily if necessary. We call this service *head of composite service*. It can be again equipped by front-end gates to match the needs of various partner service groups (Fig. 6).

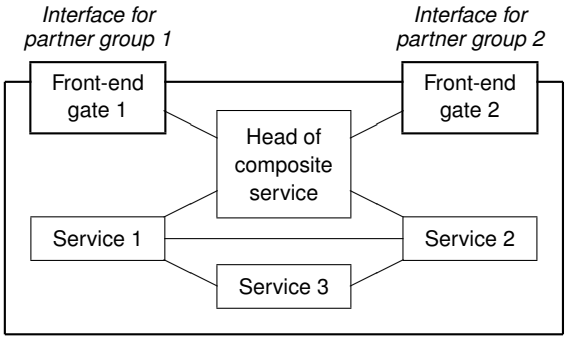

Figure 6: Composite service with head and front-end gates.

The group encapsulation simplifies error localization and system testing. The most important advantage is that the services in Fig. 6 can be again independent systems like ERP systems of business partners wrapped by their own architectural services. The architectural services are therefore a very powerful tool of the information systems integration. The organizational services together with other services form a network. The organizational services there play the role of routers.

Datastore

In structured analysis and design (Yourdon, 1988) the systems are composed from processes and data stores. The data acquisition and modification is done by processes, the data accumulation and persistence by data stores. There is a limitation that the data can get into and from data store only using some process.

If we want to integrate subsystems built using this methodology, or if we need to apply batch processing, we should equip our system with services playing the role of data stores. They substantially differ from the classic data store in maintenance and security aspects. The data stores could be also equipped with tools enabling batch as well as on-line (interactive) processing. The data stores could be used to integrate batch and interactive subsystems together (Fig. 7).

The use of batch subsystems can substantially simplify maintenance and stabilize the system.

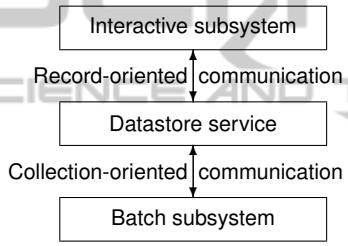

Figure 7: Composition of batch and interactive part of a system.

General Architectural Service

An architectural service can have various communication partners using various languages and protocols, a local store, supervision capabilities, and a logging ability – see Fig. 8. Architectural services described above are specialized cases of the general one.

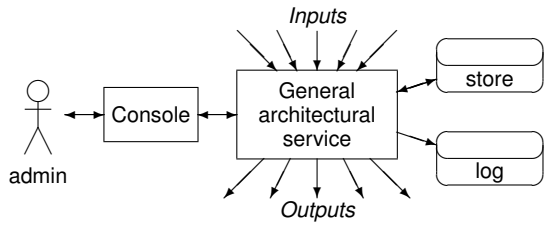

Figure 8: Generalized architectural service.

General architectural service is a very powerful generalization of the concept of connector (Taylor et al., 2009; Mikic-Rakic and Medvidovic, 2002; Bures and Plasil, 2004). It can use coarse-grained protocols, supervise and redirect messages, and powerful logging. The logs can be used by users in various situations. Examples are: business intelligence enhancements, system optimization, or business court trials.

General architectural services can further use various middleware, various message formats, and various (dynamic) possibly multipoint communication protocols.

Process Manager

Business process management can be supported by a special service – *business process manager service* or *process manager* for short – that controls execution of the business process and allows a responsible person (business process owner) to supervise and optionally modify the run of the business process.

As information system can support business processes of various types, it is reasonable to have multiple business process manager types. During a business process enactment it is possible to select a proper business process definition template that fits best the character of the business process as well as the abilities and skills of its owner.

It is crucial for emergency situations whether the process owner understands the used process definition and whether the process interface does not complicate the situation too much. The process owners must be able to use proper supervision tools. The process owners must understand the business process itself (they should be domain experts).

Encapsulation of business process supervision/control/development into dedicated services simplifies their agile definition, modification, and enhancement by users. It is a more flexible concept than the one recommended in (Erl, 2009, Chap. 8).

The business orientation of service interfaces is advantageous for the description of business processes in problem domain oriented languages. Understandable description of individual steps can directly correspond to the service requests. It can be useful for system design as well as for potential system debugging – many of the activities could be in such case easily consulted with domain experts what simplifies debugging at the problem domain level.

Screen Prototype

Screen prototype is a service of a slightly different nature. It is able to send and receive messages. The received messages are recorded and displayed to the user. The user reacts on them (either by processing/performing some activities or by writing an answer).

Screen (mock up) prototype is useful especially in the following cases:

1. It can be used to simulate the communication partners of a just developed service – it can mimic (or even replace) its communication partners.

2. It can serve as a ”manual” version of failed services.
3. It can enable manual processing of rare service requests.
4. It simplifies the integration, use, and management of batch applications.

Any standard client of our communication system (Fig. 9) can serve as a screen prototype. Otherwise it could be enhanced by supporting tools (Fig. 10) available to operator(s).

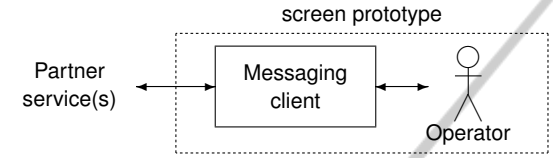

Figure 9: Simple screen prototype.

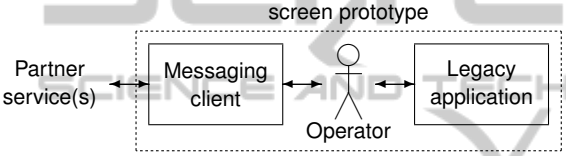

Figure 10: Screen prototype with application support.

Combined Use of Front-end Gate and Screen Prototype

If a service interface allow frequent as well as rare or unexpected requests, it is reasonable to combine advantages of front-end gate allowing efficient processing of frequent requests whereas the ones that could not be processed by the front-end gate are left for manual processing using a screen prototype and a direct interface of the application service (Fig. 11). It could be reasonable to let a human operator eventually handle some service requests that could be processed using the front-end gate. An example is the situation when standard processing could fail or lead to unwanted results. It could, for example, happen when some hot business information is available to the operator and is not encoded in the system.

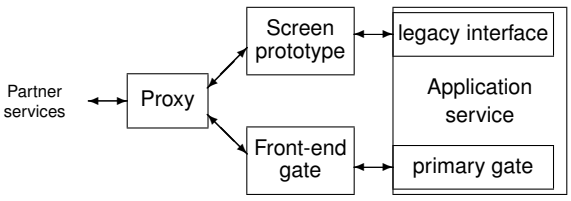

Figure 11: Application service with front-end gate and screen prototype support.

4 CONCLUSIONS

We have discussed a specific architecture – business-oriented software confederations – having very interesting and desirable properties for system development, use, and modification. It enables, if used properly, that both the development and maintenance could be based on almost identical principles and techniques. It further enables an easy involvement of managers and end-users (business people).

The architecture of software confederations enables gradual (incremental) integration and activation of services and agile modification of their collaboration. Such a development enables application of crucial elements of agile development of very large systems. It enables agility in the large with emphasis on the incremental variant of the development.

From the maintenance point of view the most important technical properties are:

- simple error or failure localization;
- the changes in different components of the system can be done using principally different tools and attitudes (changes in communication protocols, the use of logs);
- powerful information hiding in the sense of Parnas (Parnas, 1979).

The main business advantages are:

- Many operations can be effectively done by business people.
- The maintenance process is cheaper, more flexible, and can be felt as a continuation of system development.
- Collaboration between developers and business people is smooth.

A potential disadvantage could be the fact that our recommendations are not always in full agreement with current standards and design patterns from the service orientation and web service domain. One of the reasons is that the message formats should promptly reflect current user needs whereas changes of standards requires usually years. We know from our pedagogical activities and industrial projects that for people trained in object-oriented attitude it is complicated if not impossible to accept the proposal.

It is open how to combine optimally synchronous (call) and asynchronous (message passing) communication protocols. There are open security problems and the use of principles of confederations based on the virtual filling rooms being in fact generalized mailboxes accepting forms.

Our long-time experience with confederations in practical projects has shown that the confederations

are as a rule very well maintainable and reliable. Some of our confederations were used for decades. They were, however, very easy modified to match changing needs. Confederations could be very useful in the framework of SOA ecosystems (OASIS, 2012). It is open how they can be used in the frameworks like ITIL, COBIT, or ISO 20000.

REFERENCES

Aho, A. V. and Ullman, J. D. (1973). *The Theory of Parsing, Translation and Compiling*, volume II.: Compiling. Prentice-Hall, Englewood Cliffs, N.J.

Boyd, A., Pucciarelli, J., and Webster, M. (2012a). It's worse than you think: Poor document processes lead to significant business risk. [Online:] http://mds.ricoh.com/files/knowledge_center/IDC_Risk_WP_Ricoh_Eng.pdf.

Boyd, A., Pucciarelli, J., and Webster, M. (2012b). Organizational blind spot: The role of document-driven business processes in driving top-line growth. [Online:] http://mds.ricoh.com/files/knowledge_center/IDC_Revenue_WP_Ricoh_FINAL.pdf.

Bures, T. and Plasil, F. (2004). Communication style driven connector configurations. In Ramamoorthy, C. V., Lee, R., and Lee, K. W., editors, *Software Engineering Research and Applications*, volume 3026 of *Lecture Notes in Computer Science*, pages 102–116. Springer.

Erl, T. (2009). *SOA Design Patterns*. The Prentice Hall Service-oriented Computing Series From Thomas Erl. Prentice Hall Pearson Education.

Grune, D. and Jacobs, C. J. H. (2008). *Parsing Techniques*. Monograps in Computer Science. Springer, New York, USA, second edition.

Jensen, K. (1997). *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* Monographs in Theoretical Computer Science. Springer, 2nd corrected printing edition.

Krafzig, D., Slama, D., and Blanke, K. (2004). *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall Ptr.

Král, J., Demner, J., and Kostečka, V. (1979). Synchronization primitives for mass service like control software. *Polytechnica 7 (IV,1)*, pages 11–21. also in Proceedings of IFIP-IFAC 3rd SOCOCO Conference, Praha, 1979.

Král, J., Černý, J., and Dvořák, P. (1987). Technology of FMS control software development. In Menga, G. and Kempe, V., editors, *Proceedings of the Workshop on Information in Manufacturing Automation*, Dresden.

Král, J. and Žemlička, M. (2002). Component types in software conferations. In Hamza, M. H., editor, *Applied Informatics*, pages 125–130, Anaheim. ACTA Press.

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

Král, J. and Žemlička, M. (2013). Support of service systems by advanced SOA. In Lytras, M., Ruan, D., Tennyson, R., Ordonez De Pablos, P., García Penalvo, F., and Rusu, L., editors, *Information Systems, E-learning, and Knowledge Management Research*, volume 278 of *Communications in Computer and Information Science*, pages 78–88. Springer.

Mikic-Rakic, M. and Medvidovic, N. (2002). Architecture-level support for software component deployment in resource constrained environments. In *Proceedings of the IFIP/ACM Working Conference on Component Deployment*, pages 31–50, London, UK. Springer-Verlag.

Newcomer, E. and Lomow, G. (2005). *Understanding SOA with Web services*. Addison-Wesley, Upper Saddle River, NJ.

OASIS (2008). Web services for remote portlets specification v2.0. http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec.html.

OASIS (2012). Reference architecture foundation for service oriented architecture version 1.0.

Object Management Group (2011). Business process model and notation (BPMN).

Parnas, D. L. (1979). Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, 5(2):128–138.

Petri, C. A. (1962). Kommunikationen mit automaten. *Schriften der IIM*, (2).

Reynolds, W. and Antony, M. (2010). *Oracle SOA Suite 11g R1 Developer's Guide*. Packt Publishing, 2nd edition.

Robbes, R., Vidal, R., and Bastarrica, M. (2013). Are software analytics efforts worthwhile for small companies? The case of Amisoft. *IEEE Software*, 30(5):46–53.

Royce, W. W. (1970). Managing the development of large software systems. In *IEEE WESCON Proceedings*, pages 328–338. Institute of Electrical and Electronics Engineers.

Sommerville, I. (2010). *Software Engineering*. Pearson Education, 9th international edition.

Taylor, R. N., Medvidovic, N., and Dashofy, E. (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley.

W3 Consortium (1999). XSL transformations (XSLT). http://www.w3c.org/TR/xslt.

Yourdon, E. (1988). *Modern Structured Analysis*. Prentice-Hall, 2nd edition.

Žemlička, M. (2006). *Principles of Kind Parsing*. PhD thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic.