Towards Evolution Methodology for Service-Oriented Systems

Szymon Kijas and Andrzej Zalewski

Warsaw University of Technology,
Institute of Control and Computation Engineering, Warsaw, Poland
s.kijas@elka.pw.edu.pl,
a.zalewski@ia.pw.edu.pl

Abstract. Modern organisations are forced to evolve their IT systems to keep up with ever-changing business requirements. Service-Oriented Architecture addresses the challenge of boosting a system's modifiability by composing a new functionality out of existing, independent, loosely-coupled services. This makes SOA a promising design paradigm for rapidly evolving systems. However, existing development methodologies for SOA, such as IBM's SOMA, focus more on the transition from legacy non-SOA to SOA systems, and less on their subsequent evolution. This makes the development of an evolution methodology suitable for service-oriented systems an open research problem. The presented evolution methodology comprises an evolution process and an evolution documentation model. The process is compliant with a popular ISO 20000 norm. Its artefacts have been defined in terms of the evolution documentation model. The business-driven changes are documented with architectural decisions that capture changes made to the system at various levels of scope, together with their motivation. In order to facilitate the change-making process, a set of typical change scenarios has been defined. It comprises typical sequences of architectural decisions for cases of the most important changes. The entire approach is illustrated with a real-world example of an internet payment system.

1 Introduction

A system's modifiability is a primary concern for many modern organisations, which are striving to evolve their system's to meet frequently changing or emerging business requirements. Service-oriented architectures support a system's modifiability by enabling the development of a new functionality by a loose, easy-to-modify composition of existing services into new ones and by the extensive reuse of already existing services. This makes service-oriented architectures a design paradigm, which is particularly suitable for intensively evolving systems.

However, neither SOA development methodologies such as SOMA [6], SOMF [7], nor existing traceability methods [20], provide efficient and complete support for the evolution of SOA systems. This reveals a gap between the real needs of

W. Zamojski et al. (Eds.): *New Results in Dependability & Comput. Syst.*, AISC 224, pp. 255–273.
DOI: 10.1007/978-3-319-00945-2_23

SOA adopters and existing SOA development methodologies. This motivates our research on a methodology for evolving service-oriented systems presented and discussed in this paper.

The rest of the paper has been organised as follows: related work is briefly discussed in section 2, the evolution methodology is presented in section 3, its application has been illustrated on a real-world example in section 3, the contribution of this paper is discussed against the related work in section 4, and finally the outcomes and further research outlook is presented in section 5.

2 Related Work

The evolution of service-oriented systems is quite a new area of research, with rather a sparse publication record as the envisaged service-oriented world, in which services composition is the primary means of developing new functionality, is a world to come rather than the world we actually live in. In practice, service-oriented systems are currently built on top of already existing non-service-oriented ones for integration purposes [1]. Therefore, a lot of research effort has been devoted to addressing the issue of migrating legacy systems to SOA. Suitable methods can be found in, for example [6], [7], [8], [9], [10], [13]. So far, the identification and development of services to "wrap" existing functionality into services and enable interaction between systems has been the main research focus. The evolution was understood as making changes to the services, i.e. their interfaces, functionality, etc. (compare, for example [6]). The emergence of a market for third-party services and the deployment of more systems crossing organisational boundaries, possibly making their services publicly available, will change the above condition and make the evolution of business processes and service compositions a primary focus.

The research record on the maintenance and evolution of service-oriented systems is rather sparse. An idea of a transformation-driven method for evolving service-oriented systems has been sketched in [22], which seems to be, so far, the only development of this kind. Most of the research carried out so far only concerned selected evolution issues, such as changes traceability [14] (a framework for tracing changes between models of service-oriented systems), change propagation [16], [19], versioning [15], impact analysis [17], model-driven approaches to service composition, for example [18]. The research challenges in this field have been investigated in papers [1], [2], [3]. Paper [1] indicates that the development of maintenance processes is still an open research issue. Nevertheless, maintenance has been included in a post-deployment phase in [11], and has been provisioned for in the methodology presented in [12]. The evolution of services has been accounted for in the fractal process of SOMA methodology with the concept of successive iterations. In [18], the authors propose to use change management mechanisms to control the evolution of service compositions. An extensive framework for capturing architectural decisions comprising a SOA system design has been presented in [21].

Finally, let us observe that ISO 20000/ITIL [4], [5] is a set of practices for change management that has been widely accepted and adopted in industrial practice.

3 Evolution Methodology for Service-Oriented Systems

Software development methodologies, such as object-oriented or structured ones, have traditionally comprised two basic components: the development process, being a kind of a design recipe, and the supporting tools, which are used within the development process (models, notations, modelling and model analysis techniques). Our evolution methodology for service-oriented systems follows this scheme and comprises:

- Evolution process – defines a workflow, which defines how the modifications requested by business should be done in a disciplined, repeatable way, which is compliant with established industrial standards (ISO 20000/ITIL) – section 3.1;
- Evolution supporting tools – includes models used to capture the evolution of the SOA system, i.e., the model of the SOA system together with evolution documentation model, as well as techniques supporting the development of changes. The latter include: change scenarios, enriched traceability mechanism, impact analysis technique – see section 3.2.

3.1 Evolution Process

The evolution process defines a disciplined and controllable way of making numerous changes to the system. The evolution process comprises a set of instances of the modification process (fig. 1), which are initiated for every submitted Request for Change document (RFC). The modification process consists of four basic phases, which are compliant with the change management process defined in the ISO 20000:2005 standard:

– **Change assessment** – requested changes, described in the RFC, are assessed in terms of their impact (on quality attributes, SLAs, other processes, services, etc.), urgency, cost, benefits and risks;
– **Change approval** – decision makers accept or reject the submitted change. This decision is based mainly on business factors. Subsequently, the development of the approved changes is scheduled.
– **Change development and deployment** is a configurable part of the Modification process; various development processes can be applied here, e.g., agile Feature Driven Development, Scrum, XP or non-agile: waterfall, RUP. The choice should depend on the established development practices and experience of the development team.

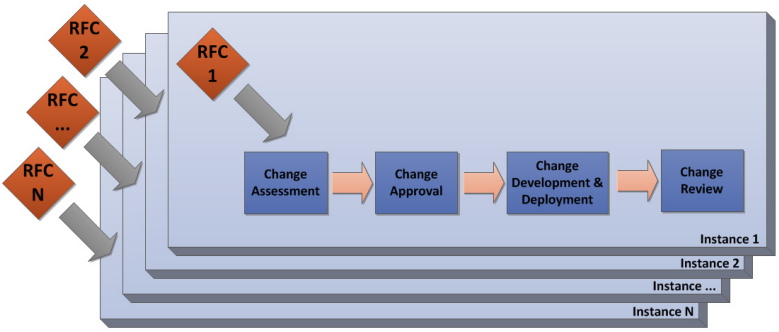


Fig. 1 Overall Structure of the Evolution Process

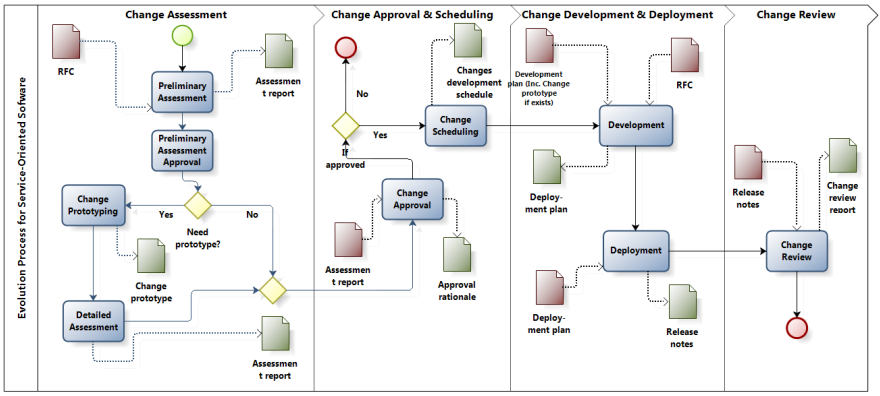


Fig. 2 Detailed workflow of the evolution process for service-oriented systems

– **Change review** is an optional phase, as required by ISO 20000. However, it should be defined whether organisation wants to include the reviews of deployed changes in its change management practices.

The detailed workflow of the modification process has been shown in fig. 2. The "Change Assessment" phase starts from a "Preliminary Assessment", in which changes described in the RFC are assessed on the basis of expert knowledge of business and system analysts in terms of their impact on functionality, quality (including Service Level Agreements), the effort needed to complete the changes and risks connected with implementing and deploying the change. The results of such an assessment are examined in a "Preliminary Assessment Approval" task, in order to verify whether they are sufficiently credible and complete in order to decide about the acceptance or rejection of the change.

"Change Prototyping" is performed if more detailed information on the impact of a change is needed in order to assess the requested change. A change prototype is a partially developed model of changes (compare section 3.2) that is supposed

to facilitate an in-depth impact analysis and will become a basis for further development if the change is approved. The final decision about the approval or rejection of changes takes place in "Change Approval and Scheduling" phase. Approved changes have to be appropriately scheduled ("Change Scheduling") to avoid conflicting changes being developed at the same time. This may also result in combining two or more changes to be developed as a single chunk. The rest of the modification process workflow seems to be self-explanatory. The artefacts of the evolution process have been defined in section 3.3.

3.2 Evolution Documentation Model

The Evolution Documentation Model consists of two basic components:

- SOA System Model (section 3.2.1) – a set of models representing the components of service-oriented systems (business processes, services, service operations and their internal logic, service compositions) at various levels of detail;
- Evolution Capturing Model (section 3.2.2) – documenting the changes introduced by the evolution steps. Such changes may concern every artefact of the SOA System Model. The evolution model provides a traceability mechanism for SOA System Models, and also facilitates impact analysis and capturing the architectural knowledge emerging during the development of changes.

3.2.1 SOA System Model

Service-oriented systems, such as presented in section III, implement one or more business processes, whose activities are supported by suitable business services. These services, in turn, comprise a number of service operations. These may be associated with the composition of a service (composed of other service operations) or developed source code. These dependencies have been reflected in the SOA System Model (fig. 3), which comprises the three layers described beneath.

Business Process Layer consists of a set of "Business Processes" supported by a service-oriented system. These BPMN models abstract from the implementation details such as service compositions, services definitions, interfaces, operations, operations' arguments, etc. Each business process is associated with a set of tasks (class "Task"), which are also included in the workflow represented in BPMN. "Task" is described using: name and description, and optionally: input and output documents (denoted by an associations with "Document" class). "Document" is described by its name and optionally: description and/or state.

Service Layer comprises a set of models that represent services used to support business processes. These models form a cascading, recursive structure as a model of a service is connected with a number of service operations, each of which can

be either an invocation of a basic (non-composed) service operation, or of a service composition, etc. The Service Layer comprises the following classes:

- **"Service"** consists of: name, set of service operations (represented as associations with "Service Operation"). Therefore, service is rather a kind of a container, or just a label for the set of its operations.
- **"Service Operation"** is an entity in which computation actually takes place. This class contains: operation name and input document – the document fed into the operation or/and output document that is the outcome of the computations (expressed as an association to "SOA Document" class).
- **"Service Composition":** model in BPMN that expresses the workflow composed of the invocations of service operations (service operations belonging to various services – internal and provided by the external providers). Service composition should be assigned to the service operation that actually provides its input and output interface.
- **"SOA Document"** contains: the name of the document and the structure of its content (i.e. XML, text or binary data). Such a document should correspond to a single "Document" from the business process layer.

Low Level Models Layer – low level, detailed models (typically in UML) and executable code. Note that these models may concern only basic services developed in-house, or being in the possession of the system's owner.

It is worth emphasising that the SOA System Model reflects the structure of real world service-oriented systems, which is particularly noticeable in the relation between services and their operations. We also assume that the tasks can be one-to-one associated with service-operations, which implement them in a service-oriented system. The same applies to the Documents and SOA Documents. This imposes certain rigour both on business analysts and SOA system designers, which is needed to make business even closer to IT.

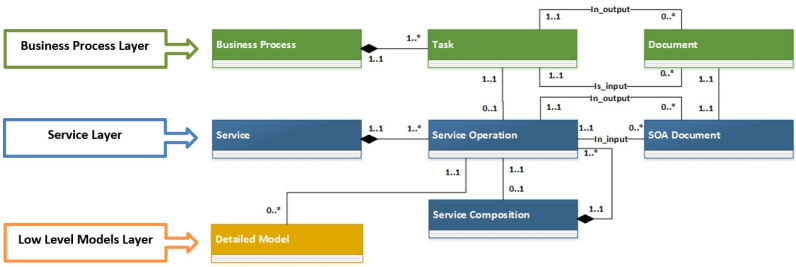


Fig. 3 Detailed architecture model of SOA system

Example. Evolution of an Internet payment system

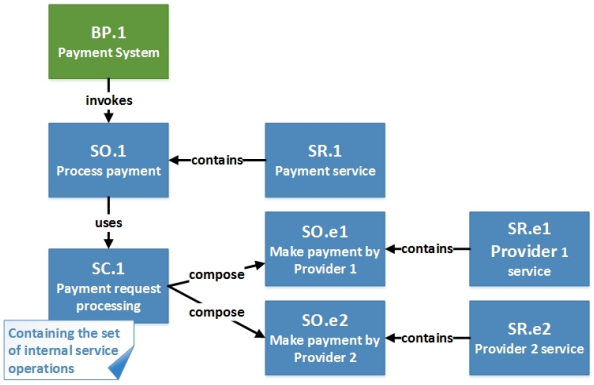


Fig. 4 SOA System Model for the "Payment System"

All the components of the Evolution Documentation Model have been illustrated on a real-world example of a portal supporting internet payments (named "Payment System"). The system comprised among others: web portal for individual customers' payments, web module for system administration and service dedicated for mass payment customers – named as "Payment service". The initial version of the system supported only two payment methods: credit or debit card payments and wire transfer payments.

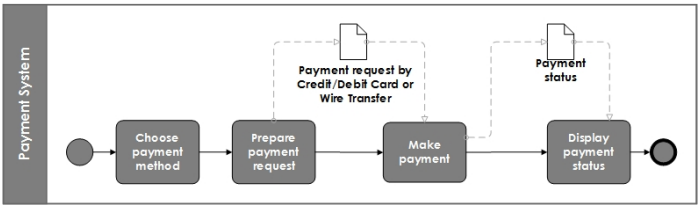


Fig. 5 Business process "Payment System"

The SOA System Model of the Payment System has been presented in fig. 4 (documents have been omitted for the clarity of the picture). It contains:

- Business Processes "Payment system" (BP.1, fig. 4): the model of a payment process implemented by the portal (fig. 5).
- Services: "Payment service" (SR.1), external services: "Provider 1 service" (SR.e1) and "Provider 1 service" (SR.e2).
- Service operations:
  - "Process payment" (SO.1), which accepts "Payment request" document and after processing the "Payment status" document is returned);

- Internal and external ("Make payment by provider 1" (SO.e1) and "Make payment by provider 2" (SO.e2)) services' operations invoked inside service composition described below.

– Service compositions: BPMN model of service composition "Payment request processing" (SC.1), which is assigned to "Process payment" service operation. It has been composed out of several service operations provided internally or externally (compare fig. 6).

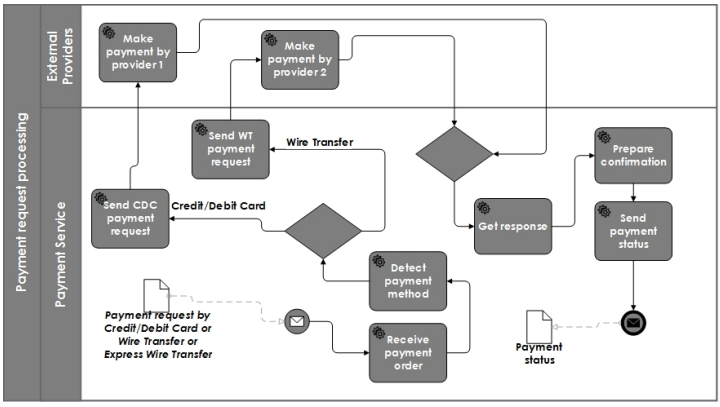


Fig. 6 Service composition "Process payment request"

The business process in fig. 5 is the "macro-flow" of the Payment system, while service composition "Payment Request Processing" defines (fig. 6) the "micro-flow" of payment processing.

3.2.2 Evolution Capturing Model

The Evolution Capturing Model (fig. 7) documents evolution as a set of "Evolution Steps". Each Evolution Step is triggered by RFC document (Request For Change), which specifies the requested change, describes its motivation, business, and if needed, technical context. The step itself comprises a cascade of architectural decisions, which capture the changes made to the models of different levels of SOA System Model. The changes made to a service-oriented system are of a cascading structure, i.e., change to a business process may force changes to services, these in turn may force changes to service compositions, which in turn may require changes to services etc. Such a cascading effect is reflected by "forces" associations.

A set of typical modification scenarios has been developed in order to facilitate the development of changes (table 1). Let us note that the change scenarios can be applied recursively.

Architectural decisions connect previous (is_input association), modified versions resulting from change's implementation (is_outcome association) as well as models' alternatives considered during change's development (is_alternative association). At the same time they provide rationale for the changes made, e.g., by justifying the choice between the connected alternatives. This concerns the following components of SOA System Model: Business Process Models, Service Models, Service Composition Models, Detailed Models.

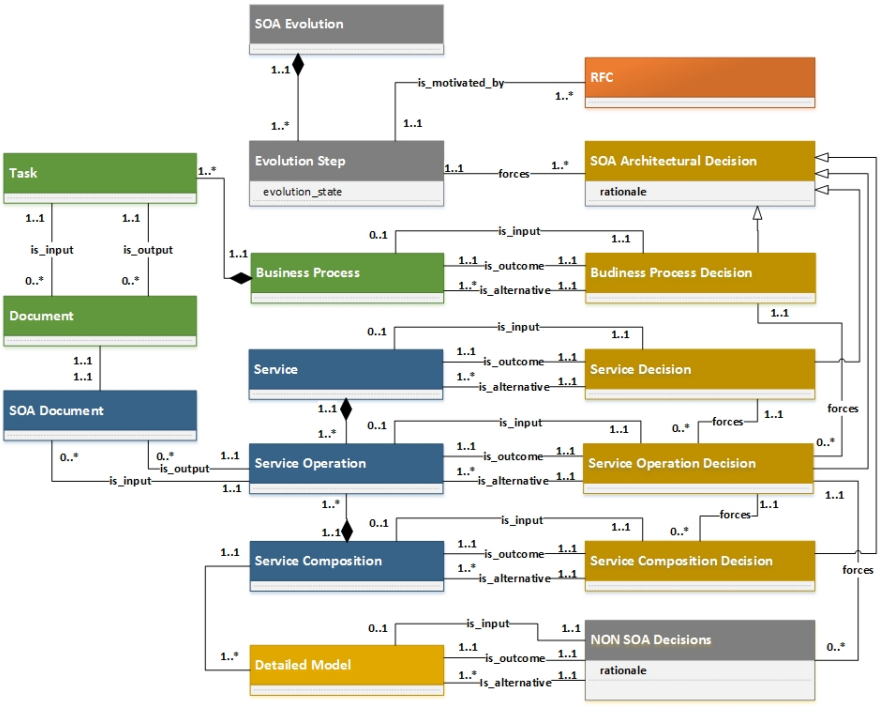


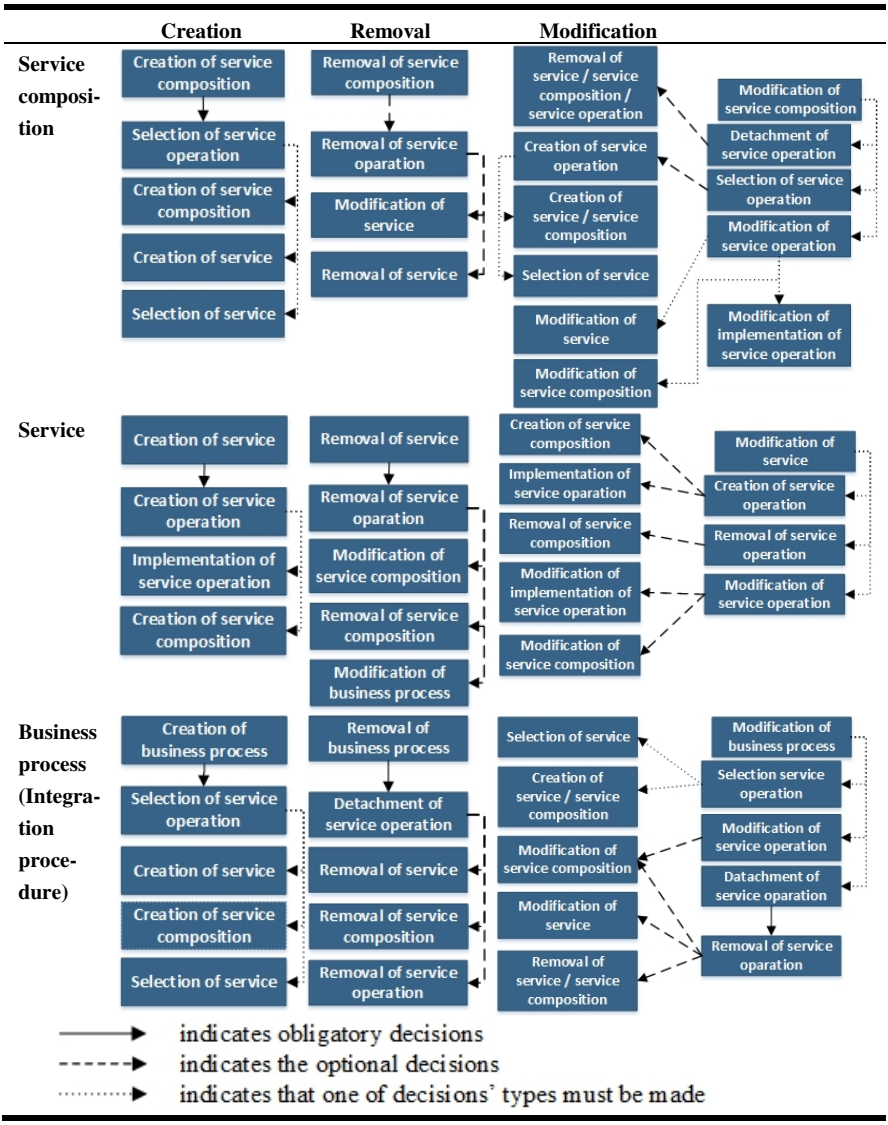
Fig. 7 Evolution Capturing Model

Example. Evolution of Internet Payment System (cont.)
Let us look back at the example to see how changes are captured using the Evolution Documentation Model presented in fig. 7.

Evolution Step No 1
Summary of RFC Document: The business expects that instant wire transfers (normally transfers are made during several communication sessions a day) will also be available.

The cascading changes necessary to implement the modifications described in RFC have been illustrated in fig. 8. The sequence of modification scenarios applied in order to develop the changes depicted in fig. 8 has been shown in fig. 9.

Table 1 The set of most popular SOA decision-making scenarios



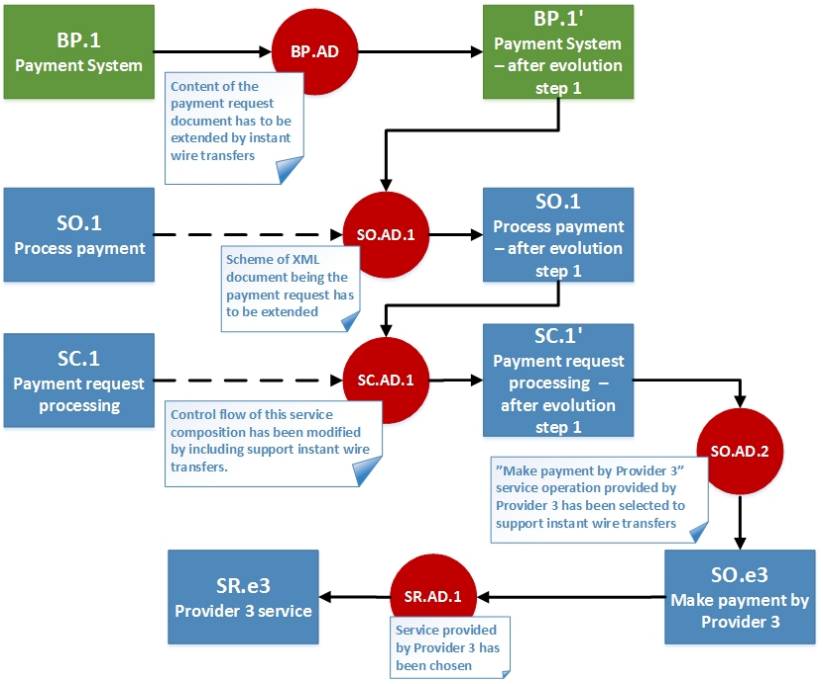


Fig. 8 The first evolution step of the payment system

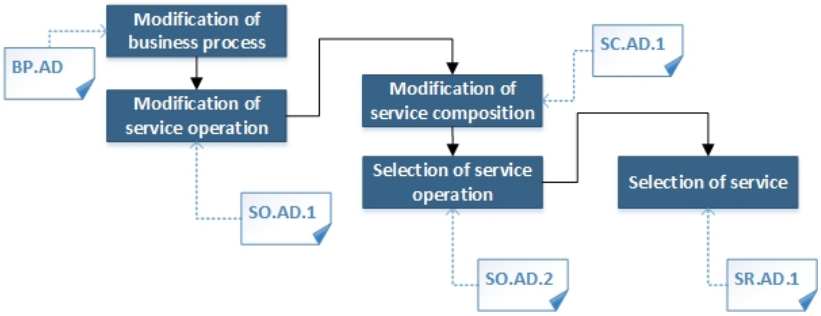


Fig. 9 Sequence of change scenarios applied to modify the system in order to support instant wire transfers

The following artefacts had to be modified:

– Business process "Payment System" – its control flow (fig. 5) remained unchanged, though, the content of the "payment order" document has been extended to include data necessary to issue an instant wire transfer.
– Service operations:

- Service operation "Process Payment" has been modified in order to support instant wire transfers – the XML scheme of the "Payment request" SOA document (corresponding to the workflow's "Payment request" document) has been extended with the information necessary for the instant wire transfers.
- Service operation "Make payment by provider 3" (SO.e3) has been added and invoked in the service composition "Process payment request".

– Services – service "Provider 3 service" (SR.e3) was added, which contains operation supporting instant transfers;
– Service composition "Process payment request" has been extended with the invocation of the service operation "Make payment by provider 3" supporting instant wire transfers (fig. 10). The composition's workflow was appropriately adjusted.

Evolution Step No. 2

Summary of RFC Document: The business expects that international instant wire transfers will also be available.

Implementation of the above changes required that a cascade of architectural decisions had to be made. These decisions capture the changes made to the models of "Internet Payment System" and their rationale. This decision making process has been illustrated in fig. 11, which extends the model developed in order to capture changes made in step No. 1. Modified versions of business process "Payment System" and service compositions "International payment request processing" can be found in fig. 12 and 13, respectively.

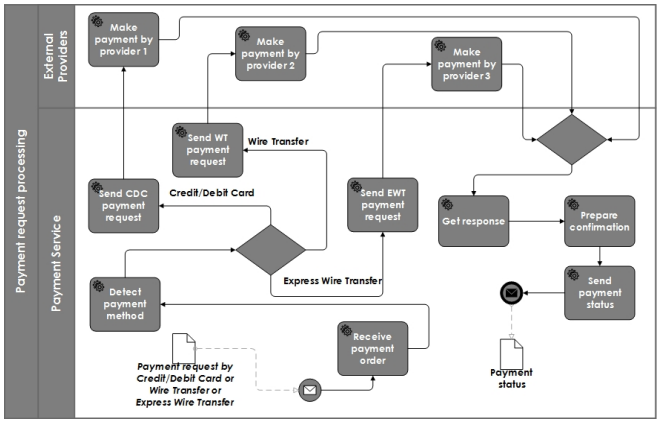


Fig. 10 Service composition "Process payment request" service operation after the first evolution step

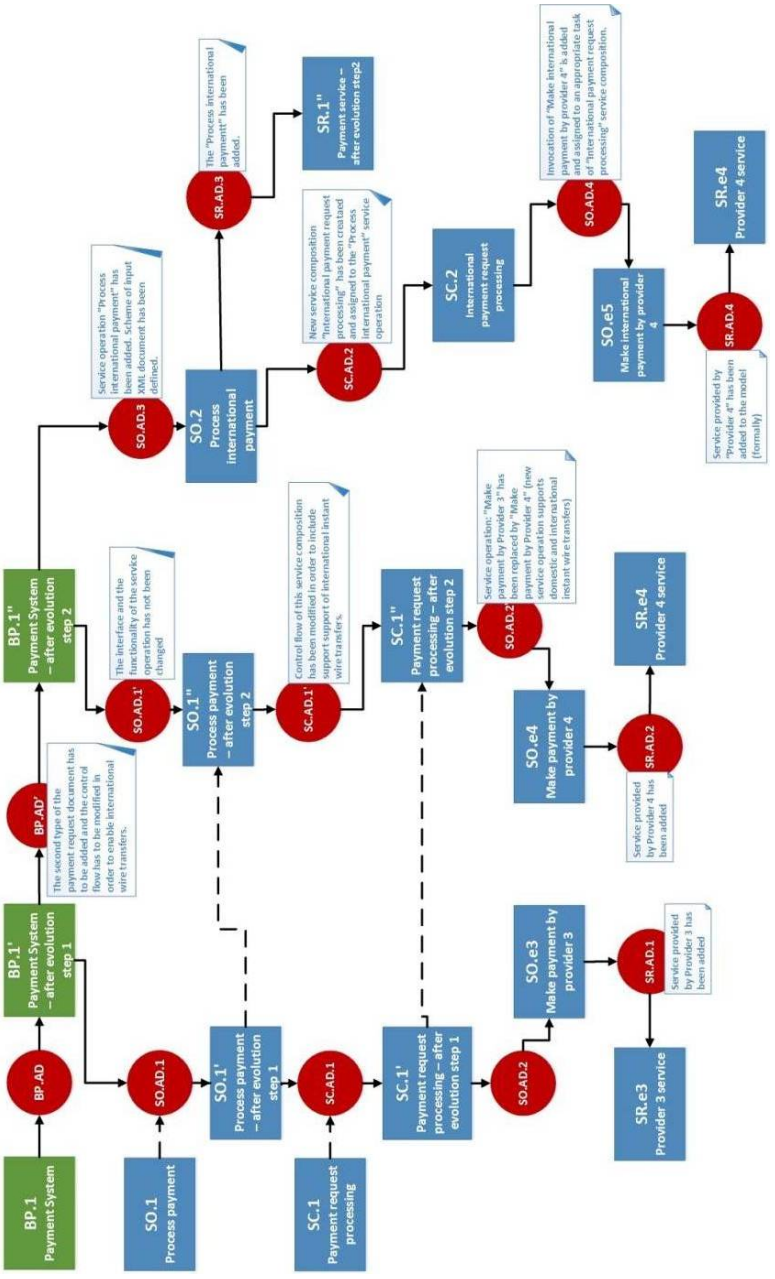


Fig. 11 Traceability model after the second evolution step

3.3 Evolution Supporting Techniques

The above example illustrates how Evolution Capturing Model can be employed so as to document the changes made to the system. Let us note that every architectural decision can not only define a traceability link between two consecutive versions of a certain model but can also be connected with the considered model's alternatives. Architectural decision includes also modification's rationale, which explains why certain changes have been made to the system. This makes it possible to understand how system has reached its current shape.

The structure of SOA System Model enables top-down impact analysis as the models potentially affected by the changes can be discovered by following the associations between higher- and lower-level (more detailed) models. The set of potentially affected models tightens as more detailed decisions are made. In such case the top-down traceability, goes from already affected models down to the possible affected subcomponents. This way the scope of changes necessary to implement a given change can be established, which should facilitate time and cost estimation. Obviously, there is a lot of space for further research in this area, which was discussed in section 4.

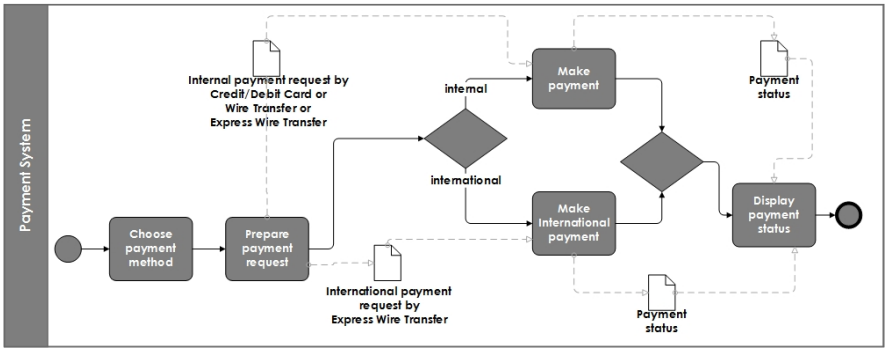


Fig. 12 Business process model "Payment system" after the second evolution step

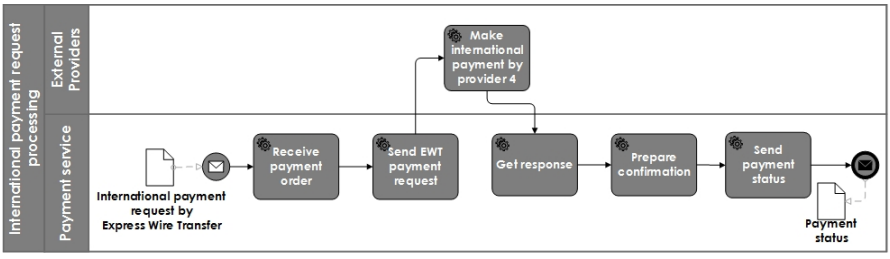


Fig. 13 Service composition "International payment request processing"

3.4 Artefacts of the Evolution Process

A detailed description of all the artefacts produced during the evolution process for service-oriented systems has been summarised in table 2.

Table 2 List of artefacts produced during the evolution process

Artefact	**Description**	**How the Evolution Capturing Model (ECM) and supports the artefacts of the evolution process artefacts**
RFC	The change is described in business or technical terms. The document also contains an explanation of the change and indications concerning its importance/priority.	RFC is included in ECM (class RFC).
Assessment report [Preliminary] or [Full]	The document includes: • scope of change: o list of business processes / service operations / service compositions modified/added/removed; • impact analysis – description of a change's impact on: o quality (including SLAs), e.g. reliability, performance, business continuity, etc.,; o list of business processes affected by the changes (e.g. requiring revision); o overlapping changes; • cost estimates, • identified risks, • attachments (other documents used for or created during the assessment process), in the case of the [Full] version of the document – change prototypes are included here.	The scope of a change can be expressed as a set of the instances of classes (Business Process, Service, Service Operation , etc.) of an SOA System Model that are subject to changes. The associations in an SOA System Model enable the impact of changes to be assessed (section 3.2.1) by identifying the artefacts that may require changes.
Change prototype	Set of business process and service composition models containing: • modified versions of existing business processes, services and service operations with the associated service compositions, • models of new processes introduced, • list of removed business processes, service compositions, service operations and services • list of detailed models subjected to change / modification / removal The above models are drafts of the assessed changes. They have not been fully developed, verified or tested.	The association *is_alternative* of ECM indicate the variants of models considered as a possible solution needed to develop a certain change. Chosen (on trial) alternatives of every modified artefact of the model comprise change prototype.

Table 2 (*continued*)

Change acceptance report	The document contains: • notes explaining the need and rationale for the approved change, • effort / cost estimated, • allocation of the cost within budget (the source of change financing); • time schedule for change development and deployment; • attachments including: RFC, Assessment reports and Change prototype.	ECM enables an analysis of the rationale of changes made to accomplish every evolution step.
Changes development schedule	A document with a schedule of all changes that have to be implemented.	The components of ECM identified as being subject to change can be used as a basis for developing a change's schedule.
Development plan	The document contains all of the information directly connected to the modification of the system: about business process models, service composition models and service models. The development plan contains the system prototype (if one exists). Additionally, this document contains all the information about detailed models and, of course, a complete set of the architectural decisions that have been made.	---
Deployment plan	The deployment plan contains installation/deployment instructions for a new release.	---
Release notes	Report on the deployment containing a list of bugs that have been corrected or are not in the developed version.	---
Change review report	Defined individually by the organisation.	---

4 Discussion

The overarching goal of our research was to develop an approach to the evolution of a service-oriented system that could be easily adopted by industry. This explains our devotion to the compliance of ISO 20000/ITIL. This is naturally an advantage of the proposed solution over the one presented in [22]. This also applies to the approaches for maintenance suggested in [6], [11], [12].

The Evolution Documentation Model provides a three dimensional traceability:

1. The history of changes made to the components of an SOA System Model (business processes, services, services' operation etc.) is captured with architectural decisions linking previous and modified versions of certain models;

2. Architectural decisions enable the motivation of changes made to the system to be captured at various levels of detail;
3. Logic of a change's development is captured with "forces" association linking changes made at various levels of detail.

The above traceability mechanism is compliant with a reference model proposed in [20], i.e. comprising satisfaction links (association between RFC and evolution step classes), evolution links (is_input and is_outcome associations between consecutive model versions and architectural decisions), rationale links (provided by architectural decisions) and dependency links (associations between the classes of the SOA System Model). In [14], the authors present a method for automatic tracing changes between models of SOA systems, both vertically (between more and less detailed models) and horizontally (between models at the same level of detail). We perceive evolution as a process of making intentional changes to the system. Admittedly, it can be facilitated with automated tools, though they cannot eliminate a conscious decision-maker – architect.

The idea of exploiting the advantages of architectural decisions for SOA systems and their evolution is becoming more and more popular. A comprehensive framework for architectural-decision making was presented in [21]. However, it does not account for the evolution of SOA systems and focuses on architectural decisions only, ignoring typical models used for SOA systems and their interrelations. Its intrinsic complexity makes it difficult to comprehend by practitioners, who have rather little time for learning elaborate methodologies. This observation became a foundation for our earlier work [23]. The proposed structure of the Evolution Capturing Model allows MAD to be employed, as a number of alternatives are associated with the architectural decisions documenting the internal logic of a single evolution step.

5 Summary and Outlook

A methodology for evolving service-oriented systems has been proposed. It comprises a disciplined evolution process and a set of models and other tools supporting the development of changes. The models have been validated on a real world example. The process's compliance with industrial standard ISO 20000 should facilitate the application of the presented approach in practice. There are obviously some missing parts of the methodology, which should become the subject of further research. Therefore, the research outlook includes:

- The development of a quality model and methods for analysing how changes impact the quality attributes;
- Supporting the development of changes with predefined model transformations applied in order to ensure that service compositions meet the quality requirements;
- The development of a software tool supporting the methodology;
- Carrying out further and more extensive validation.

Acknowledgement. This work was sponsored by the Polish Ministry of Science and Higher Education under grant number 5321/B/T02/2010/39.

References

[1] Lewis, G.A., Smith, D.B., Kontogiannis, K.: A Research Agenda for Service-Oriented Architecture (SOA): Maintenance and Evolution of Service-Oriented Systems. Technical Note, CMU/SEI-2010-TN-003 (March 2010)
[2] Lewis, G.A., Smith, D.B.: Service-Oriented Architecture and its Implications for Software Maintenance and Evolution. In: FoSM 2008, pp. s. 1–s. 10. IEEE (October 2008)
[3] Kontogiannis, K., Lewis, G.A., Smith, D.B.: The Landscape of Service-Oriented Systems: A Research Perspective for Maintenance and Reengineering. SEI (2007)
[4] ISO/IEC 20000-1:2005 and 20000-2:2005, Information technology Service management, ISO 20000-1: Specification. ISO 20000-2. Code of practice. ISO/IEC (2005)
[5] Office of Government Commerce. ITIL V3 Foundation Handbook. The Stationery Office, Updated edition (June 2009) ISBN: 978-0113311972
[6] Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K.: SOMA: A method for developing service-oriented solutions. IBM Systems Journal 47(3), s. 377–s. 396 (2008)
[7] Bell, M.: Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley Publishing (February 2008)
[8] Winter, A., Ziemann, J.: Model-Based Migration to Service-Oriented Architectures. In: Proceedings of the International Workshop on SOA Maintenance Evolution (SOAM 2007), 11th European Conference on Software Maintenance and Reengineering (CSMR 2007), Amsterdam, March 20-23. IEEE Computer Society (2007)
[9] Lewis, G., Morris, E.J., Smith, D.B., Simanta, S.: SMART: Analyzing the Reuse Potential of Legacy Components in a Service-Oriented Architecture Environment. CMU/SEI-2008-TN-008, Software Engineering Institute, Carnegie Mellon University (2008)
[10] Ziemann, J., Leyking, K., Kahl, T., Werth, D.: SOA Development Based on Enterprise Models and Existing IT Systems. In: Cunningham, P. (ed.) Exploiting the Knowledge Economy: Issues, Applications and Case Studies. IOS Press (2006)
[11] High Jr., R., Kinder, K., Graham, S.: IBMs SOA Foundation: An Architectural Introduction and Overview (November 2005)
[12] Mittal, K.: Build Your SOA, Part 1: Maturity and Methodology. IBM (May 2005)
[13] Erl, T.: SOA Design Patterns. Prentice Hall (2009) ISBN: 0136135161
[14] Sindhgatta, R., Sengupta, B.: An extensible framework for tracing model evolution in SOA solution design. In: OOPSLA Companion, pp. 647–658 (2009)
[15] Laskey, K.: Considerations for SOA Versioning. In: 2008 12th Enterprise Distributed Object Computing Conference Workshops, September 16, 2008, pp. 333–337. IEEE (2009)
[16] Dam, H.K., Ghose, A.: Supporting Change Propagation in the Maintenance and Evolution of Service-Oriented Architectures. In: 17th Asia Pacific Software Engineering Conference (APSEC) 2010, November 30-December 3, pp. 156–165. IEEE (2010)
[17] Hirzalla, M.A., Zisman, A., Cleland-Huang, J.: Using Traceability to Support SOA Impact Analysis. In: 2011 IEEE World Congress on Services (SERVICES), July 4-9, pp. 145–152. IEEE (2011)
[18] Orriëns, B., Yang, J., Papazoglou, M.P.: Model driven service composition. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 75–90. Springer, Heidelberg (2003)

[19] Ravichandar, R., Narendra, N.C., Ponnalagu, K., Gangopadhyay, D.: Morpheus: Semantics-based Incremental Change Propagation in SOA-based Solutions. In: IEEE International Conference on Services Computing, SCC 2008, July 7-11, pp. 193–201. IEEE (2008)
[20] Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Transactions on Software Engineering 27(1), 58–93 (2001)
[21] Zimmermann, O., et al.: Managing architectural decision models with dependency relations, integrity constraints, and production rules. Journal of Systems and Software 82(8), 1249–1267 (2009)
[22] Ahmad, A., Pahl, C.: Customisable transformation-driven evolution for service architectures. In: Proceedings of the European Conference on Software Maintenance and Reengineering (CSMR), pp. 373–376. IEEE Computer Society (2011)
[23] Zalewski, A., Kijas, S., Sokołowska, D.: Capturing Architecture Evolution with Maps of Architectural Decisions 2.0. In: Crnkovic, I., Gruhn, V., Book, M. (eds.) ECSA 2011. LNCS, vol. 6903, pp. 83–96. Springer, Heidelberg (2011)