

Proceedings of the First International E-Services Workshop, ICEC 03, Pittsburgh, USA

Editors: Jaap Gordijn (Free University Amsterdam, The Netherlands)
Marijn Janssen (University of Technology Delft, The Netherlands)

Table of Contents

Introduction to the First International E-Services Workshop, ICEC 03 Jaap Gordijn and Marijn Janssen.....	3
E-Services Workshop Discussion Panel William J. Ferguson, Jaap Gordijn, Marijn Janssen, René Wagenaar.....	5

Modeling E-Services

Compatibility Determination in Web Services Yunyao Li and H.V. Jagadish	7
Logic Based Approach to Web Services Discovery and Matchmaking Simona Colucci, Tommaso Di Noia, and Francesco M. Donini	15
Requirements Engineering for Service-Oriented Computing Pascal van Eck and Roel Wieringa	23
Business models for personalised real-time traffic information in cars: which route to take? Edward Faber, Timber Haaker, Harry Bouwman and Oscar Rietkerk.....	29
A Business Model Framework for E-Business Planning Chien-Chih Yu.....	38
The Configurable Nature of RealWorld Services: Analysis and Demonstration Ziv Baida, Hans Akkermans, Amaia Bernaras, Jessica Aguado, Jaap Gordijn	46

E-Government Services

Diffusion of Innovation and Citizen Adoption of E-Government Services Lemuria Carter and France Belanger	57
Exploring whether E-Government can promote McDonaldization of Governments Aby Jain	64
European Open Source Application Development Rob Peters and Ruben Wendel de Joode.....	69
Strategies for E-enforcement; Lessons Learned from two Case Studies in the Netherland Hans de Bruijn and Marieke Koopmans-van Berlo	73
Towards Implementation of E-government Services in East Africa: Content Analysis of Government Websites Janet Kaaya	82
Towards a Reference Architecture for a Virtual Business Counter Marijn Janssen and René Wagenaar.....	91

Requirements Engineering for Service-Oriented Computing: A Position Paper

Pascal van Eck^{*}
Department of Computer Science
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
vaneck@cs.utwente.nl

Roel Wieringa[†]
Department of Computer Science
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
roelw@cs.utwente.nl

ABSTRACT

Modern XML-based technologies as well as current business trends enable a move to service-oriented computing, in which functionality previously offered by application software, is now offered by the infrastructure. We argue that this enforces a new approach to requirements engineering for services, and that two different kinds of requirements engineering will emerge: requirements engineering for service consumers and requirements engineering for service providers. For the latter kind, we propose service blueprinting, which originated in marketing, as a good source to develop a requirements engineering approach to serviced-oriented computing.

1. INTRODUCTION

A current trend in computing is to provide more and more IT functionality as services to be offered by third parties. We see this trend happening inside organisations, where more and more IT functionality is provided by central departments instead of by the same departments that use this IT. We also see this trend between organisations, for instance in the form of applications being provided by application service providers (ASPs). In this paper, we argue that this trend has implications for requirements engineering, and that we expect two different kinds of requirements engineering to emerge: requirements engineering for service consumers and requirements engineering for service providers.

Recently, the term *service-oriented computing* emerged to denote the concept of computing by consuming fine-grained computational services delivered over the Internet. Service-oriented computing entails a move of functionality from ap-

plications to infrastructures. This move is not new: The company-wide database management systems that arose in the 1970s are early examples of service-oriented infrastructures. However, recent trends have accelerated the movement to service-oriented computing. First, the opening of the Web has made computing resources available world-wide to users world-wide. As has been noted by several authors, this reduces the risk of outsourcing business tasks [1]. Second, XML and associated standards such as WSDL, UDDI and DAML-S [2] provide the technical foundation for providing even small-scale functionality as a service. A service no longer needs to provide independent functionality, as provisions for service composition offered by these standards allow functionality to be combined with other functionality offered as a service. Third, mergers and acquisitions are the driving force in the business for service orientation: Functionally equivalent IT programmes in different parts of the merged organisation are taken over by shared service centers that are cheaper because of economies of scale.

The distinction between requirements engineering for service consumers and requirements engineering for service providers originates from separation of concerns. On the one hand, service consumers focus on a particular task they have to perform that is supported by an IT service. Service consumers are concerned with performing this task as effectively and efficiently as possible. In many cases, ownership and management of the hardware and software needed to support their task has no positive influence on effectiveness and efficiency. In these cases, outsourcing the IT functionality needed for the task as a service to be provided by a service provider is beneficial as it relieves service consumers of all distracting activities associated with owning and managing their own IT support.

On the other hand, the service provider, be it a central IT department in a large organisation or an independent application service providers, focusses on utilising economies of scale to provide one or more services as effectively and efficiently as possible. Economies of scale should be interpreted in a broad sense here. First, we see direct economies of scale: the additional costs for the service provider of providing a service to one more customer if there are already very many customers approaches zero. Second, we see indirect economies of scale: A large service provider, when providing a new service for the first time, is leveraged by experience gained by ongoing business and by having, on

^{*}<http://www.cs.utwente.nl/~patveck>

[†]<http://www.cs.utwente.nl/~roelw>

average, more highly skilled personnel.

This separation of concerns has consequences for both service consumers and service providers. To reach economies of scale, service providers need to offer a service portfolio that is highly standardised and stable over time. This means that service providers cannot meet each individual, business-specific function demanded by service consumers, but have to find some way to offer a set of more generic services that large groups of both current and future users may find beneficial. Service consumers have to take the current offering of services as a starting point and align their tasks and processes with these services, such that these processes can be supported with a minimal need for custom services. Requirements engineering for service-oriented computing has to take these consequences into account.

The structure of this paper is as follows. Section 2 discusses the consequences of service-oriented computing for requirements engineering and presents the two kinds of requirements engineering that we expect to emerge. In Section 3, we present service blueprinting, a well-known technique from service marketing, as an example of a technique that we think may benefit requirements engineering for service providers. Section 4 concludes the paper and discusses further research.

2. THE CONSEQUENCE OF SERVICE-ORIENTED COMPUTING FOR REQUIREMENTS ENGINEERING

2.1 Service layers

In this paper, we view an organisation, or a department within an organisation, as a stack of service layers (see Figure 1). The mission of an organisation is to deliver services to its environment¹. This mission is realised by business processes, which in turn are supported by business-specific application software. This application software is itself supported by a generic software platform (consisting, for instance, of a relational database management system, middleware, etc.). The software platform is supported by processing and network hardware that it is allocated to.

Service orientation occurs at all layers. Probably the oldest example is service orientation at the hardware layer, in the form of (national) supercomputer centres offering raw computing power as a service to universities and research institutes. We already mentioned service orientation at the platform layer in the introduction: for at least two decades, it is possible to offer storage as a service (e.g., in the form of a file server or database management system). More and more, we see that business-specific application software is offered as a service. Currently, most often this is done within one organisation, but it is also possible to outsource business-specific applications to an independent Application Service Provider (ASP). Business processes can be outsourced as well. As an example, in many financial institutions, back office processes (e.g., credit card payment processing) are outsourced to so-called shared service centres that execute these processes for more than one bank or

¹Without loss of generality, we assume that every organisation delivers services. This means that even for e.g. a manufacturer of a commodity, we view its offering as a complex service that consists of manufacturing the commodity, logistics, and pre/after sales services.

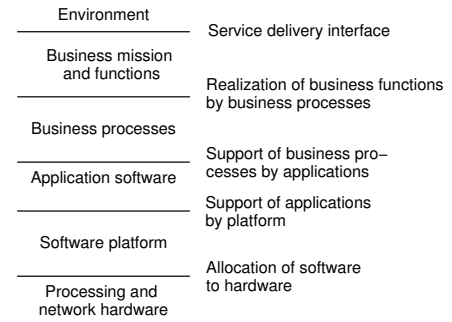


Figure 1: Service layers and interfaces.

	Centralised	Decentralised
Concentrated	No autonomy whatsoever for functional units	IT service provider owns and manages IT, but provides any service requested by departments
Deconcentrated	Decisions made at central level, but operated at many sites for reasons of performance or redundancy	Every department is completely self-sufficient

Table 1: Centralisation vs. concentration.

service brand.

2.2 Centralisation versus concentration

In IT service management, it is important to distinguish between centralisation and concentration [6]. Centralisation refers to the amount of autonomy: in a fully centralised situation, one central authority makes all decisions. In a fully decentralised situation, each functional unit of an organisation can make all IT management decisions fully autonomously. Concentration refers to ownership and management of hardware and software. In a fully concentrated situation, all hardware and software is managed by a central department in one physical location. In a deconcentrated situation, hardware and software are owned, operated and located physically within functional departments. All combinations of centralisation and concentration are possible, as illustrated in Table 1.

Centralisation/decentralisation and concentration/deconcentration can apply to any layer discussed in the previous section. In a financial institution, it is for instance possible that application software is concentrated (everyone uses a customer relationship management system that is managed by a central department), but customer relationship management processes are deconcentrated: each front-office department has its own helpdesk personnel.

Service orientation occurs in concentrated situations, where there is one department that owns and operates hardware, software, or applications, and offers it as a service to other departments. In a decentralised situation, each of

these other departments can decide for itself which services it sources from which provider, and it can negotiate with providers to customize existing services or develop new ones. In a centralised situation, it is decided by a central authority which services are offered to which department.

It is often difficult to determine whether deconcentrated IT is offered as a service. For instance, in a Dutch government branch with offices in a number of regions, file storage services are centralised but deconcentrated for reasons of performance. In each regional office, file server hardware and maintenance personnel can be found. Nevertheless, to the managers of functional departments, file storage is offered as a centralised service. In many other cases, however, deconcentrated maintenance personnel and hardware is part of the functional departments in which it is located. Whether IT policy is centralised or not, in this case the provisioning of IT functionality cannot be considered a service.

2.3 Consequences for requirements engineering

In a large organisation where IT services at one or more layers (in the sense of Section 2.1) are concentrated, there are two groups of stakeholders that deal with services:

- User groups responsible for some business function, who need IT support for this. As IT support is concentrated, this support is delivered to them in the form of a service. Therefore, we call these users *service consumers*.
- Those responsible in IT departments (where IT functionality is concentrated) for designing, implementing and providing IT services. Let us call these, in a generalized sense, *service providers*.

Service provider and service consumer are actually roles of user groups. A service consumer consumes services that support the business process this service consumer is responsible for. It may be the case that this business process is actually itself the provisioning of a service either to external clients or another department within the organisation. Thus, the service consumer is also a service provider.

Service users need to determine what IT support they need, and service providers need to determine what IT support is needed by the organisation. Although both groups try to determine what IT is needed, both groups will differ in the way they determine this, as their goals and organisational context differ. This is why we expect two different kinds of requirements engineering to emerge. In this section, we will sketch these two kinds of requirements engineering.

First, requirements engineering for service consumers. Service consumers can be found in all departments of an organisation except the IT department (where IT functionality is concentrated)². The goal of service consumers is derived from the business goal of the department in which they are located and usually concerns some part of the organisation's mission. Service consumers try to support this mission as effectively and efficiently as possible.

The primary task of requirements engineering for service consumers is to align business processes and user tasks with services offered. This means that a requirements engineer

²The IT department may itself outsource part of its own IT support. In this case, service consumers are also found in the IT department

may ask whether there is a desire for new services to be developed. But the requirements engineer may also suggest changes to the business processes or user tasks to adapt them to services currently offered. In a centralised situation, this may be the only option, as service consumers are forced to adapt to the services offered. For example, if bookkeeping operations are offered centrally as shared web services, then users have no choice but using these services. The requirements engineer, in this case, can help to adapt. In a decentralised situation, for reasons of costs, it is often better to adapt to services currently offered than to have new ones developed.

As an aside, private end users of a service (e.g., consumers at home) are considered service consumers as well. However, requirements engineering for end users in this context amounts to requirements engineering of a special class of commercial off-the-shelf (COTS) software packages for end users: packages that provide functionality by consuming services offered by third parties. Just as for service consumers in an organisation, in this case the primary task of the requirements engineer is to align end users wishes with the services offered.

Second, requirements engineering for service providers. Service providers can be found in the IT department of an organisation, or, in the case of outsourcing, in a different organisation altogether. Their goal concerns the availability of a service infrastructure for the current and future user base. Service providers need to develop and maintain this infrastructure as efficiently and effectively as possible. Therefore, they will try to provide one single generic infrastructure for all current and future users, rather than custom solutions for each specific user group.

Thus, the requirement engineer no longer constructs a specification of desired functionality needed for a concrete user task of a well-defined user group. Instead, the requirements engineer is designing an infrastructure. As is often the case with infrastructures, it is not known in advance who will use it and what the context of use is. This means that when a service is designed and implemented, a particular user group is targeted, but the individual members of the group are not known. Moreover, after the desired functionality is specified, designed and implemented, it is not handed over to the users, but remains with the service provider and is offered as a service. This means that the requirements engineer is not only responsible for specifying IT functionality, but also for designing a service that can be offered to users. In the next section, we present a technique that is used in general service design in marketing and that we think is beneficial for this aspect of requirements engineering for service providers.

Requirements engineering for service providers can be compared with for instance designing the public road system: When designed and constructed, the targeted user group consists of private individuals and transport companies. The members of this group are not known individually. The requirements for the road system are diverse and originate from the law, safety regulations, individual companies that may have conflicting interests, and the interests of private individuals who rarely give a thought about what the road system should look like. This is completely different from the development, in a traditional way, of an application that computes next year's budget in a government organization, developed for a particular government department. The ap-

plication embodies the budgeting rules of that department and has a small number of users, who, at least in theory, can all give their requirements to the application designer. The functionality offered by such an application becomes a service when it is outsourced to a system that provides this service not only to this particular department, but to a varying number of departments that are not known in advance.

Notwithstanding their differences, both kinds of requirements engineering have a number of characteristics in common. First, service level agreements (SLAs) play an equally important role in both kinds of requirements engineering. At the service consumer side, current service level agreements offered are a starting point for aligning user tasks and processes to the services offered. In a decentralised situation, a specification of a service level agreement may be one of the results of requirements engineering. This SLA is offered to a service provider as a specification of desired functionality. At the service provider side, a specification of a generic SLA is an outcome of requirements engineering that specifies a new service to be developed and offered to users. SLAs have been studied intensively (see e.g., [3, 4]).

Second, we expect that both kinds of requirements engineering will be embedded in the consumers' and provider's departments in a way that differs from traditional requirements engineering. We expect that requirements engineering will become less of a project-like kind of task (which precedes design and implementation and is at some point in time finished). Instead, requirements engineering will evolve in the direction of a continuous task, as the alignment of business processes with services offered is never finished. Requirements engineering will become a role or function in user departments and the IT department centered around continuously monitoring service level agreements to determine whether services are delivered as agreed, whether the services delivered are (still) the right ones for the task at hand (consumer side), and whether new services should be offered (provider side).

3. SERVICE BLUEPRINTING

As explained in the previous section, one of the characteristics of requirements engineering for the service provider side is that the requirements engineer is not only involved with specifying which IT functionality is needed, but is designing a service. In this section, we present service blueprinting [7], a technique that is used in general (not necessarily IT-related) service design.

Service blueprinting is a technique that describes all activities that are carried out by a service provider and its customer to deliver a service. These activities are classified in four categories:

- Service customer activities
- “Onstage” contact employee activities
- “Backstage” contact employee activities
- Support activities

Both “onstage” and “backstage” contact employees directly deal with customers (i.e., handle individual customer service delivery instances, for instance taking an order at a counter or processing a mail-in form). The difference between “onstage” and “backstage” contact employees is that “backstage” contact employees are invisible for the customer: the

customer never sees the “backstage” contact employee, or speaks with this employee. Support activities (as they are usually named in service blueprinting) are activities that are decoupled from customer contact, e.g. logistics in the case of a chain of fast-food restaurants. Despite their name, support activities in the sense of service blueprinting are often part of the core business of an organisation.

Figure 2 shows an example service blueprint. The service offered in this example is an electronic newspaper article service for which the user has to apply for a password. The top line of the figure shows the (“tangible”) evidence of service delivery. The other four lines (separated by three lines) depict the four categories of activities. Arrows between activities should be interpreted loosely as cause-and-effect relations or temporal precedence or enabling of activities. An important part of every service blueprint are activities included to resolve service delivery problems. In this example, a helpdesk is included to provide assistance in case of problems. Other such activities could for instance be activities that roll back transactions.

In constructing a service blueprint, the following design choices have to be made:

- Which service delivery activities are carried out by the consumer, i.e., what amount of self-service is involved? At service design time, it must be determined whether a customer can be expected to be able and willing to perform his or her part of the service delivery process.
- Which events occur in a service delivery process? For each event, there must be someone or something that responds to this event.
- Which part of a service delivery process is specific for a particular customer, and which part is generic?
- Which part of a service delivery process is visible for a customer (i.e., where is the line of visibility)? Activities that are visible for customers may need to be supported such that a customer is always treated with the same quality of service. This may have consequences for IT support for these activities, e.g., access to a customer relationship management system must be provided.
- What activities have to be performed in case of service delivery problems? To retain customers, it is very important to have proper procedures in case of problems. In the case of services delivered via the web, there are many different technologies to support these procedures [5].

The notation of service blueprints bears some resemblance to the notation of UML activity diagrams. Activities in a service blueprint translate to activities or wait states in an activity diagram. In activity diagrams, there is a richer notation for representing temporal relations between activities (i.e., choice, concurrency and synchronisation). The ‘line of interaction’, ‘line of visibility’ and ‘internal line of interaction’ can be represented in an activity diagram by boundaries between swimlanes.

We think that the service blueprint notation is a valuable technique in requirements engineering at the provider side³. A service blueprint makes important design choices explicit

³Service blueprinting plays a minor role in requirements en-

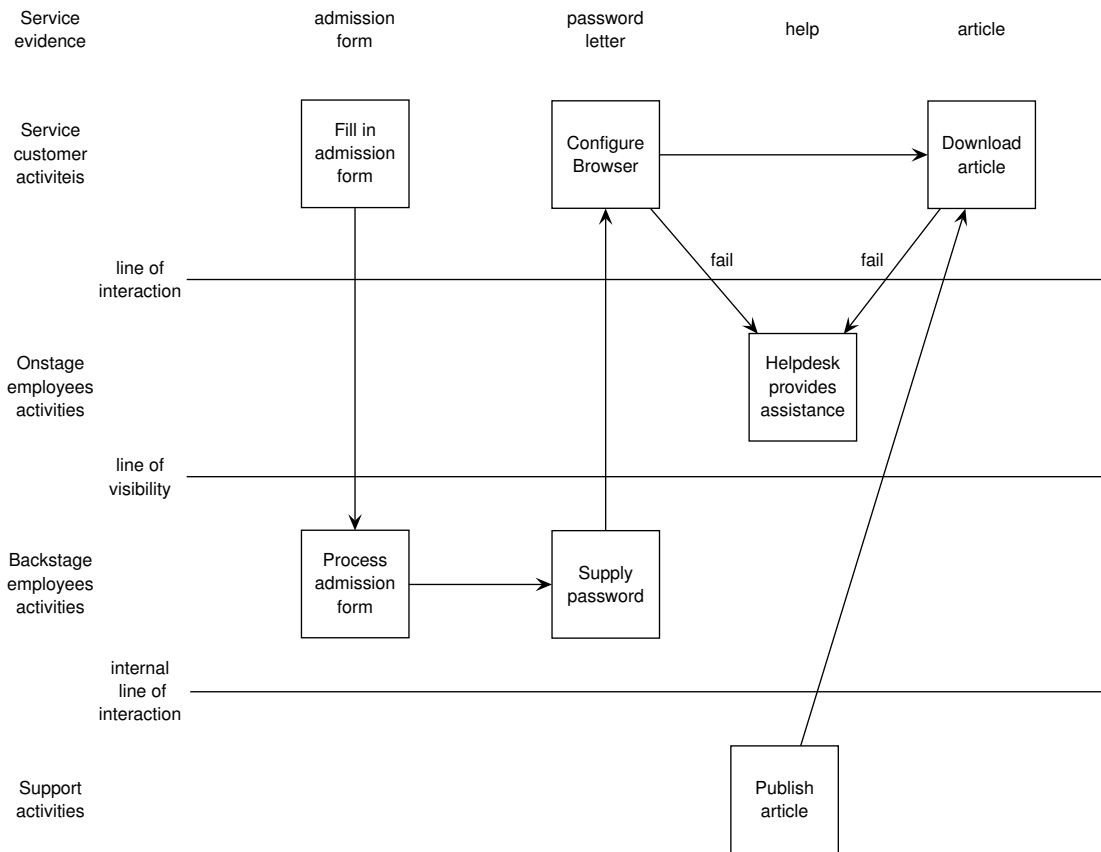


Figure 2: Example service blueprint.

without using a notation that is too formal or complex to appeal to non-IT personnel or process engineers. After service blueprints have been approved by all stakeholders, specialists can translate them into activity diagrams and add all details needed to design IT support for the service modelled in a service blueprint.

4. DISCUSSION AND CONCLUSIONS

We observe a trend to move increasingly more IT functionality from applications to IT infrastructures. We have argued that this trend will influence requirements engineering and that we expect two different kinds of requirements engineering to emerge: requirements engineering for service consumers and requirements engineering for service providers. Both forms of requirements engineering cooperate to jointly define sets of services, described in service level agreements, that supports tasks of service consumers as efficiently and effectively as possible and can be offered by central departments at a sufficiently large scale to generate significant cost savings. We have identified service blueprinting, a technique from service marketing, as a useful technique for requirements engineering for service providers.

Note that the requirements engineering task for the provider side is not the same as requirements engineering for commercial off-the-shelf software (COTS), as COTS is first and foremost a tangible product, not an (intangible) service. COTS is sold to a market and after the sale, the seller does not provide the service implemented in the COTS (but the seller may provide after-sales service in maintaining the COTS). In fact, a COTS vendor may sell its software to a service provider, who then provides it to end users as a service (in which case the end users do not recognise it as a COTS product). Requirements engineering for COTS is different as it does not have to pay attention to service delivery issues (i.e., the design choices mentioned in Section 3). Instead, requirements engineering for COTS has to pay attention to software functions that enable the COTS buyer to manage and operate the software himself.

Our approach to service-oriented computation complements the work in mainstream web services [2], which mainly has a technical focus. Whereas mainstream web services research develops for instance service registries, web service query languages and associated query processors, we analyze how to determine which services to put in a service registry (service provider side) and which queries to submit to find services that support a user group (service consumer side).

In future research, we plan to gather other ideas from the general service marketing and service management literature, apply them to service-oriented computing, and evaluate our results in case studies and consultancy work. Moreover, we plan to investigate the relation between service blueprints, UML activity diagrams and the formalisms for service description provided by standards such as DAML-S, ebXML and BPEL4WS.

Acknowledgements

The authors thank Jaap Gordijn for pointing us to the service blueprinting literature and the example, and the anonymous reviewers for a number of valuable suggestions for improvement.

5. REFERENCES

- [1] E. Clemons, S. P. Reddi, and M. Rows. The impact of information technology on the organization of economic activity: The “move to the middle” hypothesis. *Journal of Management Information Systems*, 10(2):9–35, Fall 1993.
- [2] F. Curbera, M. Duftler, R. Khalf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web - an introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):86–93, mar/apr 2002.
- [3] F. Niessink. *Perspectives on Improving Software Maintenance*. PhD thesis, Vrije Universiteit Amsterdam, Division of Mathematics and Computer Science, 2000.
- [4] F. Niessink. Perspectives on improving software maintenance. In *Proceedings of the International Conference on Software Maintenance*, pages 553–556, 2001.
- [5] N. G. Shaw and C. W. Craighead. Technology enablers to recover from failures in e-services. *Communications of the ACM*, 46(6):56–57, 2003.
- [6] T. Thiadens. Personal communication.
- [7] V. Zeithaml and M. J. Bitner. *Services Marketing, Third Edition*. McGraw-Hill, 2002.