Linda I. Terlouw and Jan L.G. Dietz

A Framework for Clarifying Service-Oriented Notions

How to Position Different Approaches

Definitions on Service-Oriented Architecture (SOA) and Service-oriented Design (SoD) are often not clear or even contradictory, which makes it hard to compare the various methodologies. We apply the Generic System Development Process (GSDP), a conceptual framework for developing systems of any kind, and specialise it for service-orientation. Using the resulting Service-Oriented Development Process, we position seven state-of-the art methodologies for service-orientation based on two criteria. One is the coverage of the system development process. The other criterion is the depth in which each of the development phases are dealt with.

1 Introduction

Most definitions of SOA (OASIS 2006; OMG 2006; The Open Group 2006) mention the notion of architecture, architectural style or paradigm, but lack a clear definition of these notions. Next, when a definition of architecture is provided, it is usually the descriptive definition, meaning that architecture is conceived as high-level models, referred to by names like 'high-level components' or 'blue prints'. Among others, The Zachman Institute for Framework Advancement (2007) uses this notion: 'Architecture is that set of design artifacts, or descriptive representations, that are relevant for describing an object, such that it can be produced to requirements as well as maintained over the period of its useful life'. Another definition that is often adopted in the context of SOA is the definition from the IEEE 1471 standard (Maier et al. 2001): 'the fundamental organisation of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution'. Apart from being unclear about the exact relationship between architecture and design, such definitions offer nothing new. The notion of fundamental organisation existed already; it is referred to by names like 'global design'.

In this article we use the prescriptive notion of architecture, i.e., we define architecture conceptually as the normative restriction of design freedom. Such a restriction is necessary and useful because the design freedom of designers, particularly in the field of software engineering is undesirable large. Practically, architecture is seen as 'a consistent and coherent set of design principles that embody general requirements' (Dietz 2008). So, we see a service oriented architecture (SOA) as a consistent and coherent set of design principles that need to be taken into account in the development process of services. Erl (2007) provides an overview of such principles. Note that we use the term 'SOA' here in a narrow sense: it refers to a particular architecture. SOA in the broad sense could better be replaced by SO (Service-Orientation), since the 'A' designates a rather vague concept.

The contribution of this paper is to present a clear terminology for SOA and SoD based on the Generic System Development Process (GSDP) (Dietz 2008; Hoogervorst and Dietz 2008). Afterwards, we position seven state-of-the-art methodologies for service-orientation based on their coverage of the development process, i.e., we look to what extent the different methodologies cover all activities within our GSDP-based process, and

on the depth with which this is done. The GSDP has been devised for the sake of understanding the mental activity of designing systems of any kind more profoundly than it is commonly the case.

The remainder of this paper is structured as follows. Section 2 presents the GSDP and its specialisation for service-orientation. In section 3 we discuss several methodologies while section 4 presents their position in relation to the GSDP. We give a brief summary of the methodologies by applying them to an insurance company case description in section 5. Finally, we draw our conclusions in section 6.

2 The Service-Oriented Development Process

The GSDP (cf. Figure 1) applies to the development of systems of any type. Whether the system is a software system, a car, or an enterprise, two different perspectives can and must always be distinguished: the function perspective and the construction perspective (Dietz 2006, 2008). The GSDP has been chosen for two reasons. First, it provides very precise definitions of very fundamental notions regarding system development, in particular the clear and useful distinction between function and construction. Second, it is the only framework the authors know of that contains a clear, appropriate, and unambiguous notion of architecture.

The GSDP defines the most basic steps in a development process. The starting point is the need by some system, called the using system (US), of a supporting system, called the object system (OS). A clear distinction between the US and the OS is often neglected, leading to blurred discussions about the functionality of the OS. From the white-box model of the US one determines the functional requirements for the OS (function design). These requirements are by nature formulated in terms of the construction and operation of the US. Consequently, they need to be fully independent on the construction of the OS. The next basic design step is to devise specifications for the construction and operation of the OS, in terms of a white-box model of the OS (construction design). For this design phase, the US may provide constructional requirements, often also called non-functional requirements.

A thorough analysis of the white-box model of the OS must guarantee that building the OS is feasible, given the available technology. In addition to the functional and constructional requirements, there may be functional and constructional principles respectively. These design principles are the operational shape of the notion of architecture, as discussed in section 1. They generally hold for a class of systems. An example of a functional principle is that man-machine dialogs must comply with some standard. An example of a constructional principle is that the applications must be component-based. Ideally the construction design phase results first into an ontological model of the OS, i.e., a white-box model that is completely independent of its implementation. Gradually this ontological model is transformed into more detailed (and more implementation dependent) white-box models, the last one being the implementation model. This process is called engineering (in the narrow sense). If the OS is a software application, then the implementation model would be the source code in some programming language. The act of implementing consists of assigning appropriate technological means to the implementation model, e.g., running the source code on an appropriate platform.

2.1 Service Design

Both the black-box model and the white-box model of an application are relevant to service-orientation: the first for specifying and using services and the second for building or changing services. The design phase consists of two steps: function design and construction design. Function design results into a black-box model. The first, and highest level, white-box model that is produced during the construction design is the

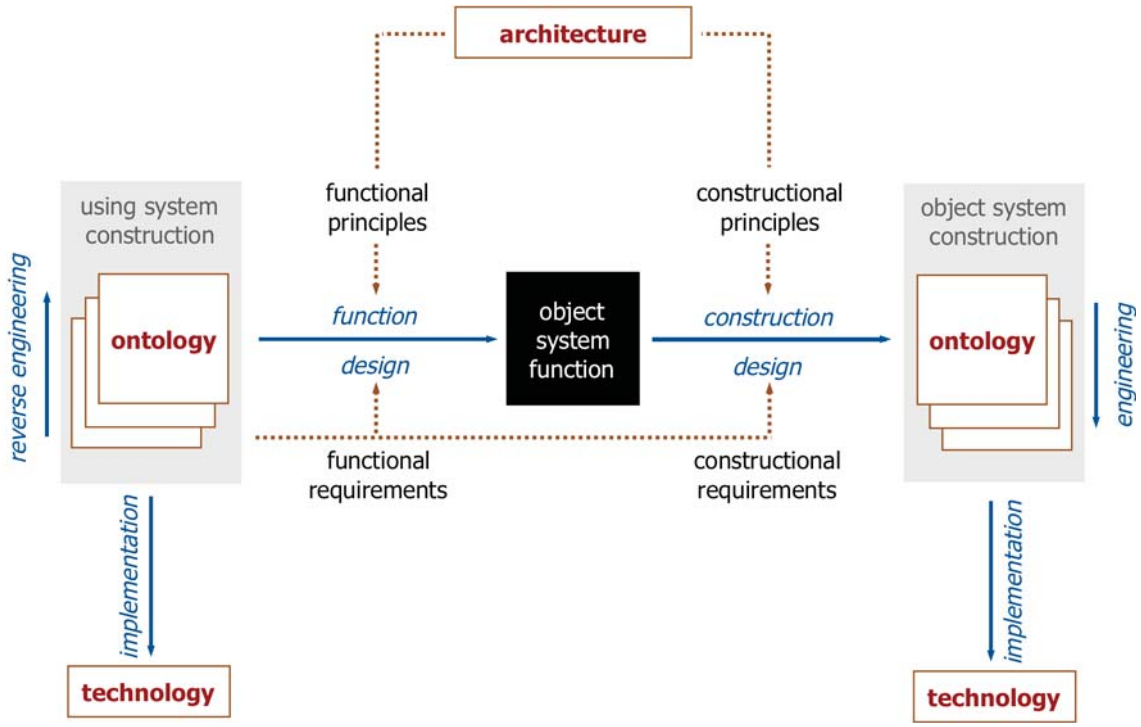


Figure 1: The generic system development process (Dietz 2006)

ontology or ontological model of the OS. The term ontology originates from the field of philosophy having the meaning 'essence or existence'. In our context we define ontology as follows (Dietz 2006): the ontology of a system is the understanding of the system's operation that is fully independent of the way in which it is or might be implemented. Applying this to services we see two phases in SoD: Service-oriented Function Design (SoFD) and Service-oriented Construction Design (SoCD). SoFD deals with identifying and specifying the function of a service. Service identification is in our view the first step in SoFD. It deals with identifying candidate services. The central question is 'What services are required in the scope of the enterprise (or enterprise network)?'. The process of service specification (including Quality-of-Service requirements) is part of SoFD as well, since it specifies the external behaviour of a service without caring about its internals. Once the specification is available one can design the construction of a service.

SoCD thus starts with producing the ontological model of the service. A service needs to be constructed in such a way that it conforms to the constructional principles of the applicable architecture. A common classification of services is that of atomic and composite services. For the service consumer there is no notable difference in which way the service provider constructs a service. Composite services can be constructed by, for instance, orchestration (e.g., BPEL) or assembly (SCA Open SOA 2007). Since there is no language for modelling services at the highest (ontological) level, one might consider a (high-level) class diagram to do this job for object-oriented modelling.

2.2 Service Implementation

The implementation of the object system is the assignment of technology to the lowest-level white- box model (the implementation model). Hoogervorst and Dietz (2008) use technology in the broadest possible meaning, e.g., human

beings, software systems, electronic machines. Service Implementation (SI) deals with mapping the implementation model to computer systems. This is the phase in which the services are deployed. Most enterprises have at least two test environments to deploy to after the service has been developed: a test environment for verification (checking whether the service matches the specifications) and one for validation (checking whether the service is useful to potential service consumers). So both the developers and the end-users have their own, independent test environment to suit their own purposes. Both tests can lead to repeated execution of previous steps in the design process and redeployment to the test environment. Finally, the service is deployed to the production environment.

2.3 Service-Oriented Architecture

As said before, the architecture of a system consists of the design principles that have been or are going to be applied in designing the system. As stated in section 1 we define SOA as a consistent and coherent set of design principles that need to be taken into account in the development process of services. The functional principles deal with the external function and behaviour of a service, e.g., 'A service must directly support an activity within a business process' or 'A service must be compliant with the Dutch law'. The constructional principles deal with the internal construction and operation of the service, e.g., 'A service must be constructed with commercial-of-the-shelf components' or A composite service must be created by using BPEL. Table 1 summarises the terminology explained in this section.

3 Methodologies

This section presents the state-of-the-art methodologies for service-orientation. We selected seven methodologies that are described in scientific papers or books (i.e., not in confidential business documents). In our eyes these are the service-oriented methodologies that are discussed in most detail, but we do not imply that this is a limitative list. These methodologies appear to be either generic, i.e., having a broad scope and focusing mainly on which steps to take and not how to execute the steps, or specialised, i.e., focusing on a specific part and covering it in-depth. In this section we apply the terminology as used by the authors of the papers describing the methodologies. In section 4 we will map the design steps of these methodologies to those of the GSDP.

3.1 SOAF

The consulting company Infosys Technologies developed the Service Oriented Architecture Framework (SOAF) (Erradi et al. 2006). The goal of SOAF is to devise a systematic and repeatable process for implementing SOA. SOAF combines top-down modelling of business processes with the bottom-up analysis of the existing applications. The authors do not mention the origins of SOAF explicitly, except for past experiences with SOA in practice.

SOAF describes a number of activities that need to be performed for migrating to a service-oriented environment. Also, it describes the inputs required by a certain activity and the deliverables created by it. The activities are grouped into several phases. From the total of 19 deliverables, 6 are marked being most important. Not all of the activities are discussed in the article. The phases of SOAF are:

1. Information Elicitation
2. Services Identification and Matching
3. Services Realisation
4. Roadmap and Planning

3.2 P&H

Papazoglou and Van den Heuvel (University of Tilburg) propose the SoD and Development methodology (Papazoglou and Heuvel 2006), which from now we call P&H. The authors not only provide an overview of the activities required to migrate to a service-oriented environment, but they

Table 1: Terminology derived from applying GSDP to services

Term	Meaning	End result
SOA	a coherent set of design principles that need to be taken into account in development process of services	all services in scope conforming to the same principles
Service-Oriented Function Design	deals with the design of the function of the service	the service specification of an identified service
Service-Oriented Construction Design (including Service Engineering)	deals with the design of the highest-level white-box model and with the decomposition of the highest-level white-box model to the lowest-level white-box model	the full design of the internals of the service
Service Implementation	deals with the mapping of the lowest-level white-box model to technology, i.e., the deployment of services	the deployed service

also explain some of the architectural principles that apply to the functional and constructional design of services, e.g., functional service cohesion, communicational service cohesion, identity coupling, and communication protocol coupling. This methodology is partly based on other related development models, such as the Rational Unified Process (RUP), Component Based Development and BPM. The authors' starting point is the idea that a good methodology for service-orientation is based on an iterative and incremental process and that it concentrates on business processes. The phases of P&H are:

1. Planning
2. Analysis
3. Service Design
4. Service Construction
5. Service Test
6. Service Provisioning
7. Service Deployment
8. Service Execution
9. Service Management and Monitoring

3.3 SOMA

IBM proposes Service-Oriented Modeling and Architecture (SOMA) as a methodology. The first version was presented in 2004. SOMA is an extension to existing IBM analysis and design methodologies, including the Global Services Method, a methodology used by IBM Global Services, and the Rational Unified Process, a method for software development widely adopted in industry. SOMA started out as an IBM Global Services Method. Meanwhile IBM was working on SOA specific extensions to the Rational Unified Process. In 2006 the people working on both initiatives joined forces. In 2006 an overview of the method (Arsanjani and Allam 2006), consisting of three phases, was published. Recently (2008), a more detailed article (Arsanjani et al. 2008) on SOMA has been published. Two additional phases were added. As far as we know, SOMA has no strong theoretical basis. Instead it is based on a large number of project expe-

riences over 2002–2004 (Arsanjani 2006). SOMA distinguishes between the following five major phases:

1. Identification
2. Specification
3. Realisation
4. Implementation
5. Deployment, Monitoring, and Management

3.4 BCI3D

Business Component Identification 3D (BCI3D) (Albani and Dietz 2005) is a methodology for identifying business components, i.e., software components that directly support the business processes, using clustering algorithms. Also, it identifies the services by which the business components interact. The calls between these business components are regarded as the highest level services an organisation requires. BCI3D comprises the following phases:

1. Enterprise Ontology modelling
2. Relationship Weight Definition
3. Application of Clustering Algorithm

3.5 Business Element Approach

McGovern et al. (2006) describe the Business Element Approach in their book 'Enterprise Service Oriented Architectures'. Its main goal is to make it possible to trace changes to business processes or rules quickly and unambiguously to one or more specific components. The Business Element Approach has it roots in Component Based Development methodologies. The following phases belong to the Business Element Approach:

1. Requirements Definition.
2. Business Element Analysis
3. Mapping to Components

3.6 Goal-driven approach

The goal-driven approach (Levi and Arsanjani 2002) derives services from business goals. The services are depicted in a Goal Service Graph and they are allocated to enterprise components. This Goal Service Graph provides traceability of IT services to business goals. These enterprise components are identified by clustering highly interdependent (coupled) use cases. One of the authors of the article on the goal-driven approach also co-authored the article on SOMA we used in our paper. The goal-driven approach finds its basis in the world of Component Based Development and Object-Oriented analysis and design methods. Also, it builds upon formal grammar specification to define domain-specific languages. SOMA (Arsanjani et al. 2008) mentions this method as one of the possible methods for service identification. The goal-driven approach consists of:

1. Domain Decomposition
2. Subsystem Analysis
3. Creation Goal Model
4. Service Allocation
5. Specification of Enterprise Components
6. Structuring Enterprise Components

3.7 SMART

The Service-Oriented Migration and Reuse Technique (SMART) (Lewis et al. 2005) is a methodology that describes in detail on how to construct services from legacy systems. SMART takes into account that in practice it is often not easy to construct services from legacy systems. Since almost no organisation has the luxury to build up its entire IT-environment from scratch, it is important to recognise the risk involved in migrating to a service-oriented environment. According to SMART an organisation needs to thoroughly assess the capabilities of its legacy systems and carefully analyse the risk of migrating. SMART is developed by the Software Engineering Institute (SEI). The US DoD sponsored the work. SMART was derived from the Options Analysis for Reengineering (OAR) method (Smith et al. 2002). The phases of SMART are (Lewis et al. 2005):

1. Establish Stakeholder Context
2. Describe Existing Capabilities

3. Analyse Gap between Service-Based State and Existing Capabilities
4. Develop strategy for service migration

4 Positioning the Methodologies

In this section we investigate to what extent the different methodologies cover the GSDP-based service-oriented development process. Table 2 exhibits the mapping of seven methodologies, in alphabetic order, to the service-oriented development process derived from the GSDP. The vertical axis consists of the phases of the methodologies as described in section 3. The horizontal axis comprises the phases of the specialised version of the GSDP.

BCI3D deals with SoFD only. It identifies services working from business models but it does not deal with how the business components offering these services are constructed. The same thing holds for the specialised methodologies Business Element Approach and Goal Driven Approach. The only specialised methodology that has a different focus is SMART. This methodology focuses on the whole SoCD phase (including Service Engineering (SE)), since it looks at how legacy systems can implement the identified candidate services. It assumes other methodologies are applied for acquiring these candidate services. The SMART phase of establishing stakeholder context is not really a design activity; it takes place before the actual development process starts.

Looking at the generic methodologies, we see that P&H and SOMA have the broadest scopes. SOAF does not cover the SI phase. Let us have a closer look at these three methodologies.

SOAF's information elicitation and its service identification and matching phase both deal with service function design. The latter, however, also deals with service construction design as the ease of composition is also taken into account. The SOAF realisation phase places more emphasis on SoCD and the decomposition of the highest-level white-box model into lower-level white-box model, viz. SE. The planning of the deployment of the service takes place in the SOAF Roadmap and Planning phase. The phase does not deal with the deployment itself.

In the planning phase of P&H some decisions about the function and construction of the services are taken like the business processes they need to support and the existing systems that can be used for implementation. The thorough function design of the services takes place in the analysis and service design phases, after which the construction takes place in the service construction phase. Testing belongs both to the SoFD and to the SoCD phase as it is concerned with testing the function as well as the construction (links between the components) of a service. The methodology explicitly mentions the service deployment step (the SI phase).

In the paper from 2006 (Arsanjani and Allam 2006) SOMA consists of three phases: identification, specification, and realisation. In later work (Arsanjani et al. 2008) two phases are added: (i) implementation, and deployment, and (ii) monitoring, and management. However, these two phases are not discussed in detail in the new paper (they are said to be out of scope). SOMA's identification phase as well as its specification phase both map to the SoFD phase of the GSDP. The SOMA realisation phase as well as its implementation phase deal with the construction of services (SoCD). Finally, the deployment, monitoring, and management phase deals with SI.

In both SOMA and P&H the authors speak of activities like 'monitoring' and 'management'. However, we do not consider these activities to be part of the development process for services as defined in the GSDP. These activities deal with the operation of the system instead of with its development. The amount of detail in which the phases of the generic methodologies are described is the largest in P&H. Also, this methodology does not only deal with the steps that need to be performed in the service design process, but also emphasises the principles applied during the

design process. These six design principles focus on minimising the coupling between services.

5 A Brief Summary of the Methodologies

In this section we analyse the methodologies using an insurance company case. For this company, called Protector, we provide examples on how the different methodologies would work. The business models from these case are derived from a real-world insurance company. For the sake of simplicity, we only show part of the models.

5.1 Introduction

Protector, an insurance company, sells three types of life insurance products. The first type of product, the term life insurance, protects the beneficiaries of a policy from the financial damage they suffer in case the insured dies during the policy term. The second type, pension insurance, can be seen as an insurance that protects the insured from a large income loss if he reaches his pension age or that protects his life partner and young children from large income loss after the insured (would have) reached his pension age in case of the insured's death. The third type, the capital sum insurance, is an insurance for building up capital. When the end date of the policy is reached, then the benefit will be paid in a single payment. This product type is, for instance, suitable for saving money to pay off a mortgage. Protector offers multiple products of each type.

Figure 2 depicts the relationship between the relevant information objects in UML notation. When a certain insured person would cause a high risk for Protector, because for example the insured amount is high, then the policies are reinsured by a reinsurer. This means that a part of the insured amount is insured by the reinsurer in order to spread the risk.

Protector has two main business goals for the next 5 years: (i) standardising their product portfolio and (ii) increasing customer satisfaction. Currently, the company tends to create new products for almost every corporate client, which results in huge policy management problems. The company wants to improve its product management process and restrict the freedom of sales employees in defining their own new products. Customer satisfaction is low because it takes a long time to process changes to a policy. Also, clients are complaining about the lack of support of self service through the Internet. The client has to contact the company by phone or mail. Table 3 exhibits the six main software systems of Protector. Protector is considering replacement of the legacy systems.

5.2 SoFD in BCI3D

Recalling subsection 2.1, we decompose the SoFD phase into service identification and service specification. They respectively deal with determining what services are required and with documenting the external behaviour of a service. BCI3D has the Enterprise Ontology as a starting point. Figure 3 shows parts of the so-called Actor Transaction Diagram (ATD) and Transaction Result Table (TRT) of the Enterprise Ontology of Protector.

The ATD depicts the actor roles and the transaction kinds. For example, CA03 is the initiator of transaction kind T05, of which A05 is the executor (indicated by the small black colored box). A05 is also initiator of T26 and T27. The TRT lists the transaction kinds and their result types (in which a word in iltalics denotes a variable). BCI3D uses the state model (an information object model) of the Enterprise Ontology. The state model is denoted in an ORM-based language. It combines the information objects as defined in Figure 2 with the result types of transactions. For instance, the object 'Policy' would be associated with the result type of the transaction 'Policy Quotation', i.e., policy pol is quoted. It would go too far to explain the complete DEMO methodology that is used for creating this models. Its main

Table 2: Classification of methodologies for service-orientation

Methodology	Phase	SoFD	SoCD	SI
BCI3D	Enterprise Ontology modelling	O	-	-
	Relationship Weight Definition	O	-	-
	Application of Clustering Algorithm	O	-	-
Business Element Approach	Requirements Definition	O	-	-
	Business Element Analysis	O	-	-
	Mapping to Components	O	-	-
Goal Driven Approach	Domain Decomposition	O	-	-
	Subsystem Analysis	O	-	-
	Creation Goal Model	O	-	-
	Service Allocation	O	-	-
	Specification of Enterprise Components	O	-	-
	Structuring Enterprise Components	O	-	-
SOAF	Information Elicitation	O	-	-
	Services Identification and Matching	O	O	-
	Services Realisation	-	O	
	Roadmap and Planning	-	-	-
P&H	Planning	-	-	-
	Analysis	O	-	-
	Service Design	O	-	-
	Service Construction	-	O	-
	Service Test	O	O	-
	Service Provisioning	O	-	-
	Service Deployment	-	-	O
	Service Execution	-	-	-
	Service Management and Monitoring	-	-	-
SOMA	Identification	O	-	-
	Specification	O	-	-
	Realisation	-	O	-
	Implementation	-	O	-
	Deployment, Monitoring, and Management	-	-	O
SMART	Establish stakeholder context	-	-	-
	Describe existing capabilities	-	O	-
	Describe the future service-based state	-	O	-
	Analyse the gap	-	O	-
	Develop strategy for service migration	-	O	-

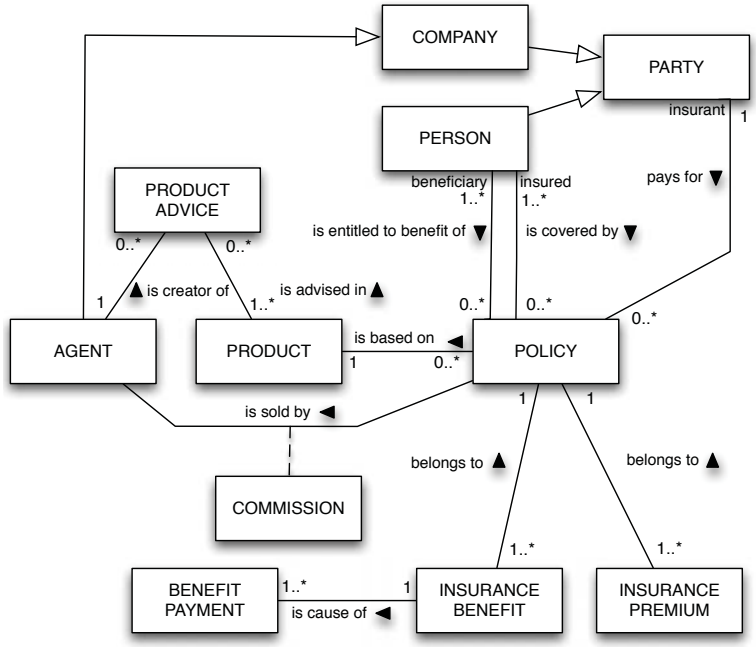


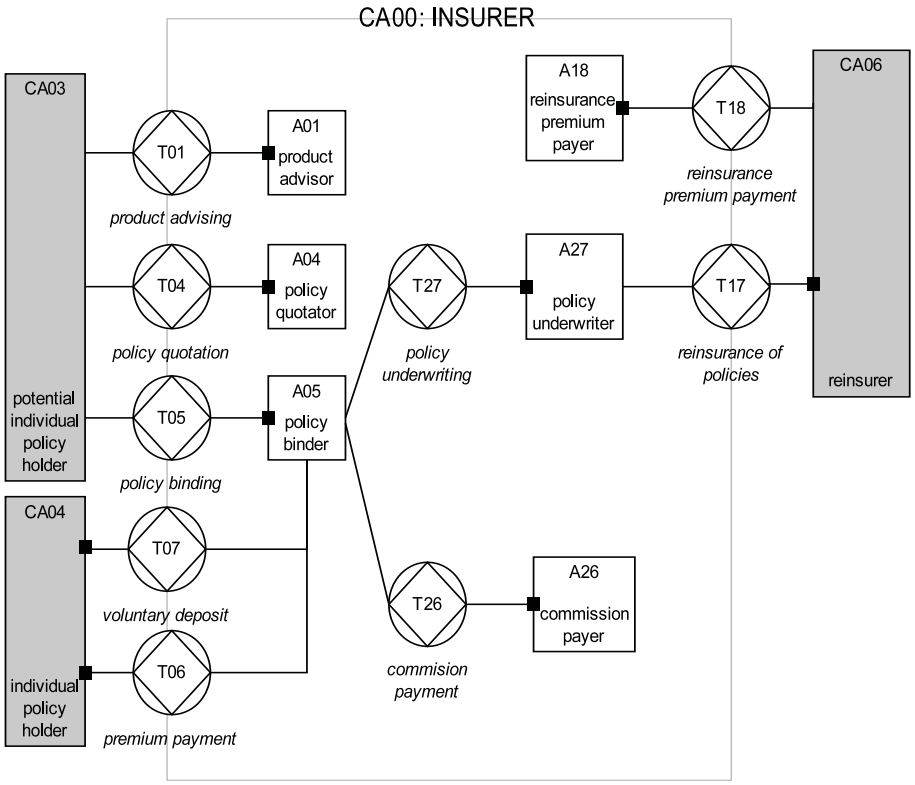
Figure 2: Information Object Diagram for Protector

idea is to make two types of abstractions in business models. The central notion of the enterprise ontology is the transaction, which is a pattern of coordination acts required to achieve a production result. Capturing these acts together in one transaction, because they constitute a universal pattern, leads to the first reduction in the size of the models. The second reduction of the size is achieved by considering only the so-called ontological production acts, leaving out infological and datalogical ones. The models most relevant for BCI3D are the interaction model, of which the ATD is a notation, and the state model.

Based on architectural principles the user of the method can set weights to the relationship between transactions, between transactions and information objects, and between information objects. An example of such a principle is 'information objects with a parent-child relationship should reside in one component'. This principle would result in a very high weight for the relationship between information objects having a parent-child relationship. Let us assume that payment management and policy management are recognised as components. In that case 'Commission Payment' could be a service offered by the payment management component to the policy management component, because this policy management component needs to issue a commission payment to an agent after a policy binding. BCI3D does not deal with service specification.

5.3 SoFD in the Business Element Approach

The Business Element Approach does not require the business goals as input. It only uses the information objects model and the process model. The criteria for identifying Resource Business Elements (RBE's) are whether they are 'real' and 'independent'. 'Real' means that a Subject Matter Expert (SME) both uses and understands it. An 'independent' resource is one that somebody can talk about without first saying to what it belongs. As our information object model is constructed together with SME's all objects in it are 'real'. The policy object is an 'independent' resource, the benefit payment, insurance benefit, and in-



Transaction	Result type
T01 Product advising	product advice *adv* is created
T04 Policy quotation	policy *pol* is quoted
T05 Policy binding	policy *pol* is bound
T06 Premium payment	premium is paid for policy *pol* for premium period *per*
T07 Voluntary deposit	voluntary deposit is made for policy *pol*
T17 Reinsurance of policies	policy collection *pco* is reinsured for period *per*
T18 Reinsurance premium payment	reinsurance premium is paid for policy collection *pco* for period *per*
T26 Commission payment	commission *com* is paid
T27 Policy underwriting	underwriting for policy *pol* has been done

Figure 3: ATD and TRT of Protector

surance premium objects are not. In the example given by the authors, roles are also explicitly modeled as objects, which is not the case in our model. We can, therefore, also consider the following objects as auxiliary resource elements: insurant, insured, and beneficiary. This leads to the RBE depicted in Figure 4.

Now the services are identified by process decomposition. We look at the top-level immediate steps defined in the process model (i.e., a step 'that is required to complete as soon as possible,

Table 3: Software Systems of Protector

System	Purpose
Siebel	A CRM for managing data related to reinsurers, insurants, beneficiaries, and insured persons
InterAgent	A custom-made system for managing data related agents, the advices they provide, and the commission they receive
OmniPayment	A custom-made system for paying insurant benefits to beneficiaries, for paying reinsurance premium to the reinsurer, and for pay commission to agents
DirectPolicy	A newly acquired commercial system suitable for pension and capital sum policy administration. This system can be used for self-service over the internet.
DinoPolicy	A legacy system for pension policy administration that can only be used by employees of Protector
TermPolicy	A legacy system for term life and capital sum policies that can only be used by employees of Protector

and whose intermediate states are of no concern to the business in that they are not required to be remembered after the process has completed'). After that, the subsidiary immediate steps are defined. Finally, the services are grouped in SBE's by grouping the steps by RBE. In our case this lead to the results exhibited in Table 4.

A DBE is 'a grouping of Service and Resource Business Elements that together deliver a business solution to a business problem, and which provides services to requesters'. In our case example DBE's are Product Management, which contains the Product Service SBE and the Product RBE, and Policy Management, which contains the Policy Service SBE and the Policy RBE, the Person RBE, and the Company RBE.

The method does not deal with service specification.

5.4 SoFD in the Goal-Driven Approach

When applying the goal-driven approach, we first define the component boundaries in terms of business processes. A possible business process division is:

- Product Portfolio Management
- Customer Relationship Management
- Quotation
- Policy Management
- Reinsurance

In the E-bazaar example (Levi and Arsanjani 2002) given by the authors, these major business process areas are broken down into lower level business processes. E.g., product portfolio management can be decomposed as follows: Product Portfolio Management = {Product Development, Product Addition, Product Removal}.

Once the structure of the components is identified through domain composition and subsystem analysis, we can identify services using Goal Service Graphs. Working from the two business goals mentioned in the case description, we construct the Goal Service Graph as depicted in Figure 5. We decompose these business goals until we get a set of services that will achieve a certain goal.

In the service allocation step we assign services to the components. We can map the 'Calculate risk for product' and the 'Check for duplicate products' services to the 'Product Portfolio Management' component. In the process of developing a new product, we need to calculate the risk involved in a certain product (mostly by statistical analysis). The second service is used in the process of adding new products to the portfolio to decide whether it is sensible to start working on a new product.

Table 4: Two of the SBE's of Protector

SBE	Service	Subsidiary Immediate Steps
Policy Service	Request Quotation	Validate Client Data Record Client Data Validate Policy Related Data Record Policy Related Data Send Receival Confirmation Letter
	Change Policy	Validate Requested Changes Calculate Risk Involved With Changes .....
	.....	.....
Product Service	Add Product to Portfolio	Check for Duplicate Products Insert Product
	Remove Product from Portfolio	
	.....	.....
	Develop Product	Make Initial Product Design Calculate risk for product Make Product Final .....
	.....	.....

When looking at the second step of SoFD (service specification), the authors speak about Enterprise Component Specification. They address the following three key ingredients of such a specification: services (interfaces), contracts, and manners. Services (interfaces) specify 'what capabilities the component provides to support the business goals and processes, what it requires to do so, and an abstract specification of the design of how the services realise goals'. Contracts refer to the specification of pre- and post-conditions of each service and the sum total of which services are provided and required by the component. Manners specify 'how the component in a given state should behave within a given context; which subset of rules to check once an event has been triggered'.

5.5 SoFD in SOAF

SOAF positions service identification as an iterative process for arriving at an optimal set of services. The authors propose a hybrid approach combining top-down domain decomposition along with bottom-up application portfolio analysis. The activities result in a list of candidate services that further needs to be rationalised and consolidated. The activities are not discussed in sufficient detail for enabling us to apply the ideas to our case.

Service specification is called service description in SOAF. The following aspects are considered to be part of the service specification: the service interfaces and data types of exchanges messages, the behavioural model of the service including the supported message exchange patterns (e.g., one-way / notification or request-response), the

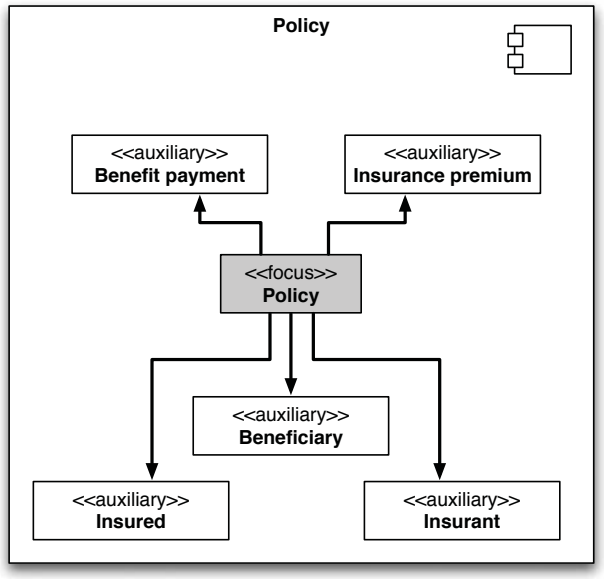


Figure 4: One of the RBE's of Protector

1. Standardise product portfolio
   a) Migrate expired policies to new standardised products based policies
      - *ConvertPolicy*
   b) Apply strong product management
      - *Check for duplicate products*
      - *Calculate risk for product*
2. Increase customer satisfaction
   a) Decrease change processing time
      - *Search for policy*
      - *Calculate effects of change*
   b) Enable self service through the Internet
      - *Request quotation*
      - *Request policy*
      - *Change policy*

Figure 5: Goal Service Graph

supported conversation and temporal aspects of interacting with the service, and the service policy. The service 'BindPolicy' might comprise the specification elements exhibited in Table 5.

Table 5: Example of part of a service specification in SOAF

Bind Policy	
Exchange pattern	Request-response
Input	Id of policy data object
Output	DateTime of the moment of binding
Preconditions	Policy *pol* has been created Policy *pol* has been underwritten
Effect	Policy *pol* is bound
Availability	99,5%
Required protocols	XML Encryption XML Signature

5.6 SoFD in P&H

P&H deals with service identification, though not in much detail. We were not able to identify services from our case using this methodology. The authors, quoting Johnston (2005), mention the following service specification elements: structural specification, behavioural specification, and policy specification. Structural specification refers to how the interface, specified in the Web Service Definition Language (WSDL), is structured, e.g., messages, port types, and operations. The paper does not discuss how this relates to service-oriented environment that do not use Web services. The behavioural specification deals with understanding the effects and side effects of the services (called 'operations' by the authors) and the semantics of input and output messages. In our case the behavioural specification of the service 'Bind Policy' could state how the consumer cannot be used if the policy is not created and underwritten first. The policy specification denotes policy assertions and constraints on the service. These assertions may cover security, manageability, etc. In our case policy specifications might be that the Bind Policy services (i) uses XML Encryption and (ii) is available 99,5% of the time.

5.7 SoFD in SOMA

For the process of service identification SOMA mentions three main complementary techniques:

goal service modelling, domain decomposition, and existing asset analysis. For the domain decomposition the authors refer to a technique called variation-oriented analysis and design (VOAD). Existing asset analysis takes a bottom-up approach; it looks at the existing application portfolio and other assets and standards that may be used in identifying good candidate services. We already explained the goal-driven approach in subsection 5.4. In the paper we used as a source for writing this subsection, the mentioned goal service modelling and domain decomposition techniques were combined. When applying the existing asset analysis approach, we could suggest candidate services like 'Pay Reinsurance Premium' offered by the OmniPayment application and 'Create Term Life Policy' offered by the TermPolicy application.

SOMA addresses the importance of service specification. The specification describes the following aspects of a service (called 'operation' in the paper): non-functional requirements, input, output, and error messages. Also, the authors mention a context diagram, i.e., 'the service ecosystem for a group of related services illustrating service consumers and service providers and indicating the flow of messages between services and the various underlying systems that implement them'. In our eyes the underlying systems that implement the services would not belong to the service specification.

5.8 SoCD in SOAF

SOAF does not describe in detail what steps to take for SoCD, but it does present a taxonomy of service realisation approaches. The precise steps depend on the approach taken. The taxonomy distinguishes between non-invasive and invasive approaches. Non-invasive service enablement is defined as 'a tactical approach to align existing systems to business needs through wrapping by using new layers of flexible technologies such as Enterprise Application Integration (EAI) solutions, messaging tools, and recently with standardised interfaces using web services'. Invasive transformation is defined as 'a strategic approach that aims to revitalise and streamline the application portfolio to ease maintenance, extension, and interoperability'. In our Protector case this means the following. Let's say we need a service for 'Policy Quotation'. A non-invasive approach would be constructing a service by screen scraping a legacy application like DinoPolicy or by creating a Java DataBase Connectivity (JDBC) adapter for an existing database to get some additional required data. In an invasive approach, one would reconsider the current application landscape. Examples are to reengineer Dinopolicy to include the additional data or to buy a COTS system that comprises the complete functionality.

5.9 SoCD in P&H

The authors discuss the following possibilities for creating a service: (i) starting from scratch, (ii) use an existing application to construct the service, and (iii) compose a new service from other services. They discuss green-field development, top-down development, bottom-up development, and meet-in-the-middle development. Green-field development assumes that first the service is constructed and then the service interface is generated. Top-down development starts with a service interface after which the service is constructed. Bottom-up development assumes the service interface is developed from an existing application. Meet-in-the-middle refers to the mapping of an already existing service interface (for which the service is already constructed) to a new service definition.

5.10 SoCD in SOMA

In the paper from 2006 (Arsanjani and Allam 2006) SOMA consists of three phases: identification, specification, and realisation. In later work (Arsanjani et al. 2008) two phases are added: (i) implementation and deployment, and (ii) monitoring, and management. However, these two phases are not discussed in detail in the new pa-

per (they are said to be out of scope). We can conclude from the paper that the following activities in the Realisation phase belong to SoCD: Refine and detail components, Establish realisation decisions, Perform technical feasibility exploration. Also, the construct, generate and assemble services step of the Implementation phase belongs to SoCD. During the technical feasibility step prototypes are built that exercise the realisation decisions and that have the highest potential impact and risk to the non-functional requirements. In our case we could imagine that an important decision for a Policy Quotation Service is security, since the client has to deliver confidential data to Protector. The result of the Construct, generate and assemble service step is verified in the steps Execute unit test and Execute integration and System test (also belonging to the implementation phase).

5.11 SoCD in SMART

SMART is a technique that 'helps organisations analyse their legacy systems to determine whether their functionality, or subsets of it, can reasonably be exposed as services in an SOA'. It describes in a large amount of detail which steps are required to analyse legacy systems. Some of the techniques used are code analysis and architecture reconstruction. Let us assume OmniPayment is an application written in an object oriented language. Example results from the legacy analysis are exhibited in Table 6.

Table 6: Example results from the legacy analysis in SMART

	OmniPayment
# of services covered	6
Total # of lines of code	13,284
# of classes affected	22
# of lines of code affected	5,392
Level of difficulty	Medium
Level of risk	Low
Use of coding guidelines	Strictly followed
Use of design patterns	Many violations found

5.12 SI in P&H

The authors make a clear distinction between development and deployment. They define deployment as 'rolling out new processes to all the participants, including other enterprises, applications and other processes' (a process being a BPEL process). The phase is not discussed in detail, so we cannot apply it to our case.

5.13 SI in SOMA

As we already mentioned, the articles we used as sources do not describe the deployment, monitoring, and management phase. Though the concept of service deployment is mentioned, the authors do not give any suggestions on how to deal with it. Therefore, we were not able to apply it to our case.

6 Conclusions

Currently, most methodologies for service-orientation fail to make a clear distinction between the notions of SOA and SoD. The main contributions of this paper are first, an elucidation of these notions based on the Generic System Development Process (GSDP), and second, a positioning of several methodologies for service-orientation. SOA has been defined as a consistent and coherent set of design principles that need to be taken into account in the development of services. We have defined SoD as the design part of a development process, according to the GSDP. It consists of producing successive conceptual models of the object system under consideration, starting from the ontological model. In SoD, this object system is a service.

We introduced the following phases in the development process for services: Service-Oriented Function Design, Service-Oriented Construction

Design (including Service Engineering), and Service Implementation. Using this high-level distinction of phases we have determined the scopes of the investigated methodologies for service-orientation.

Two of them cover the whole development process, namely P&H and SOMA. However, they do not describe all phases very thoroughly. Some of the specialised methodologies provide an in-depth contribution to specific steps of the development process, like BCI3D for the Service-Oriented Function Design phase (including service identification), and SMART for the Service-Oriented Construction Design phase.

We applied all the methodologies on the same small case. From these exercises we draw the next conclusions. The strength of BCI3D is that it is based on the well-defined notions of business process and business object of DEMO. Another strength of BCI3D is that it uses algorithms for the identification of services, thus requiring minimal human effort and ensuring objectivity. What is still lacking, however, are guidelines for choosing weights. Like it is the case for BCI3D, the strength of the Business Element Approach is that the identification of services is based on business (information) objects and business processes. However, there is no clear notion of business process nor of information object. Clustering requires human insight, thus it is more subjective than BCI3D. Applying the goal-driven approach to the Protector case yielded that it is even more subjective than the Business Element Approach. In particular, we think it's a naive and erroneous idea that the identification of services can directly be based on the goals of an enterprise. One then neglects the intermediate level of the construction of the using system (as defined in the GSDP). When applying SOAF, SOMA, as well as P&H to the case, we were left with some questions. All three methodologies prescribe the steps to be executed, but not clearly and completely. In contrast, SMART is very precise in how to perform the service construction phase. Because SMART is the only method we found that deals with service construction in detail, we were unable to compare it thoroughly.

All in all, the GSDP has been very helpful in discussing the coverage of the investigated methodologies, and to elucidate and sharpen the core notions in service-orientation. Next, the application of the methodologies to the same case has shown the differences in the depth in which the activities are described. None of methodologies combines full coverage and full depth.

References

Albani A., Dietz J. (2005) Basic Notions Regarding Business Processes and Supporting Information Systems. In: Requirements Engineering 10(3), pp. 175–183

Arsanjani A., Ghosh S., Allam A., Abdollah T., Gariapathy S., Holley K. (2008) SOMA: a method for developing service-oriented solutions. In: IBM Syst. J. 47(3), pp. 377–396

Arsanjani A. (2006) Best Practices in Service-oriented Architecture. http://www.ibm.com/developerworks/blogs/page/-AliArsanjani. Last Access: 2009-10-03

Arsanjani A., Allam A. (2006) Service-Oriented Modeling and Architecture for Realization of an SOA. In: SCC '06: Proceedings of the IEEE International Conference on Services Computing. IEEE Computer Society, Washington, DC, USA, p. 521

Dietz J. (2006) Enterprise ontology - understanding the essence of organizational operation. In: Enterprise Information Systems VII, pp. 19–30

Dietz J. (2008) Architecture - Building strategy in design. Academic Service, Amersfoort, The Netherlands

Erl T. (2007) SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl). Prentice Hall PTR, Upper Saddle River, NJ, USA

Erradi A., Anand S., Kulkarni N. N. (2006) SOAF: An Architectural Framework for Service Definition and Realization. In: IEEE SCC. IEEE

Computer Society, pp. 151–158
Hoogervorst J., Dietz J. (2008) Enterprise Architecture in Enterprise Engineering. In: Enterprise Modelling and Information Systems Architectures 3(1), pp. 3–13
Johnston S. (2005) Modeling service-oriented solutions. http://www.ibm.com/developerworks/rational/library/jul05/johnston/. Last Access: 2009-10-03
Levi K., Arsanjani A. (2002) A goal-driven approach to enterprise component identification and specification. In: Communications of the ACM 45(10), pp. 45–52
Lewis G. et al. (2005) SMART: The Service-Oriented Migration and Reuse Technique. Technical Note, CMU/SEI-2005-TN-029. http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn029.pdf
Maier M. W., Emery D., Hilliard R. (2001) Software Architecture: Introducing IEEE Standard 1471. In: Computer 34(4), pp. 107–109
McGovern J., Sims O., Jain A., Little M. (2006) Enterprise Service Oriented Architectures: Concepts, Challenges, Recommendations. Springer-Verlag New York, Inc., Secaucus, NJ, USA
OASIS (2006) Reference Model for Service Oriented Architecture, Committee Draft 1.0.. http://www.oasis-open.org/committees/download.php/16587/wd-soa-rm-cd1ED.pdf. Last Access: 2009-10-03
OMG (2006) Service Oriented Architecture Special Interest Group (SIG). http://soa.omg.org/.
Open SOA (2007) Service Component Architecture. http://www.osoa.org/display/Main.
Papazoglou M., van den Heuvel W.-J. (2006) Service-oriented design and development methodology. In: International Journal of Web Engineering and Technology 2006 2(4), pp. 412–442
Smith D. B., Brien L. O., Bergey J. (2002) Using the OAR Method for Mining Components for a Product Line. In: SPLC 2: Proceedings of the Second International Conference on Software Product Lines. Springer-Verlag, London, UK, pp. 316–327
The Open Group (2006) Service Oriented Architecture. http://www.opengroup.org/projects/soa/.
The Zachman Institute for Framework Advancement (2007) Enterprise Architecture: A Framework

Linda I. Terlouw
ICRIS B.V.
Martinus Nijhoffhove 2
Nieuwegein
The Netherlands
linda.terlouw@icris.nl

Jan L.G. Dietz
Department of Information Systems
Delft University of Technology
Mekelweg 4
Delft
The Netherlands
j.l.g.dietz@tudelft.nl