Engineering Virtual Domain-Specific Service Platforms

Specific Targeted Research Project: FP7-ICT-2009-5 / 257483

Report on State of the Art in Service Platform Design, Adaptation, Deployment and Monitoring

Abstract

This document represents the report on the state of the art in service platform design, adaptation, deployment, and monitoring. The document adopts a wide spectrum to survey the main achievements and proposals in the fields of service-roeitned systems, requirements elicitation, adaptation, product line engineering, deployment, and monitoring techniques to pave the ground to the proposals and techniques developed within the project.

Document ID:	INDENICA – D1.1
Deliverable Number:	D1.1
Work Package:	1
Type:	Deliverable
Dissemination Level:	Public
Status:	draft
Version:	2.8
Date:	2011-09-30
Contributing Partners:	PDM, SAP, SIE, TUV, SUH, UNIVIE, TEL

Project Start Date: October 1st 2010, Duration: 36 months

Version History

0.1	01. Jan 2011	Initial version
1.0	01 Feb 2011	First contributions added
1.1	01 Mar 2011	First integrated version
1.2	18 Mar 2011	Polishing and integration
2.0	21 Mar 2011	Consolidated version
2.1	22 Mar 2011	Minor changes
2.2	23 Mar 2011	Bug fixing
2.3	25 Mar 2011	References integrated
2.4	29 Mar 2011	Proof reading and final version
2.5	15 Sep 2011	Refinement of Section 2 (SAP)
2.5.1	15 Sep 2011	Refinement of Section 2.5.3, 4.3, 5.1 (by SUH)
2.6	23 Sep 2011	Refinement of Section 2.6, 4.5 (by UNIVIE)
2.7	30 Sep 2011	Refinement of Section 6 (by TEL)
2.8	30 Sep 2011	Proof reading and new final version (by PDM)

Document Properties

The spell checking language for this document is set to UK English.

1 Introduction

Service technologies have been gaining more and more attention over the last years. They started as glue among heterogeneous parties and flexible infrastructures at application level, and they are now the foundations of the “cloud revolution”: everything is seen and accessible as a service. Platforms and infrastructures offered and exploited as services are becoming common and many big players (e.g., Microsoft, Google, and IBM) are already offering complete and interesting solutions. More and more platforms ---both publicly available and with limited visibility--- will be available in the next years, but at the same time all these different solutions are posing new and compelling problems.

We need to clearly define what a platform is. INDENICA defines a service platform as an assembly of infrastructure assets, like communication middleware or databases, and platform services that together constitute the interface and programming model for application service development. These platforms should be able to cope with user requirements. In these days, one can think of developing a new platform to cope with particular needs, but we are quickly moving towards scenarios where a significant set of alternative platforms exists, and new solutions have to be conceived by tailoring and combining them.

The functional and non-functional properties of a service platform and its interface must vary with the requirements of a domain. This is why INDENICA proposes the idea of domain-specific service platform as the result of matching users needs, implicit domain assumptions, and available solutions to support applications that require the integration of services across platform boundaries. Adaptation becomes a key issue, and product line principles can be an interesting enabler to cope with all the different alternatives in a consistent way.

This means that the problem of conceiving, designing, and realizing a domain-specific service platform is multi-faceted and must be tackled under different viewpoints. This document identifies some interesting aspects (angles), and describes the relevant state of the art to provide an initial assessment of available solutions, identify significant challenges, and pave the ground to the new solutions proposed by the project.

The document starts with the definition of some key concepts and a brief survey of well-known commercial solutions. Then, it addresses the problem under four parallels threads: requirements elicitation, design principles, adaptation, governance and deployment, and monitoring and runtime supervision. Although platforms are gaining more and more interest, a large amount of existing solutions still address the problem at application level, but this trend should be reverted in the next few years. The last part of the document tries to move a step forward the pure state of the art and concludes the document by identifies some challenges that belong to the research field, and that as such INDENICA will address over the next years.

1.1 Structure of this document

The rest of this document is organized as follows. Section 2 introduces some definitions of the key concepts behind service platforms, and briefly describes some well-known solutions. Section 3 addresses the problem of requirements elicitation and also the business considerations behind the design/selection of a service platform. Section 4 is about design solutions, architectures, and models. Section 5 presents the main results as for implementing adaptation, governance, and deployment. Finally, Section 6 considers the runtime supervision of service platforms and surveys solutions for monitoring their execution, assessing their quality of service parameters, and enforcing service level agreements. Section 7 concludes the document and identifies some challenges for the project.

2 Background

This section introduces the main concepts behind the engineering of virtual domain-specific platforms.

2.1 Service

There is a plethora of definitions for service. The FP7 project NEXOF-RA has set up a glossary of terms, which are related to service orientation and platforms[1]. They have a very short and pragmatic definition of service: *"A service is an action performed by one Entity that fulfils a request of another Entity"* whereas entity is defined as "An entity is a person or organization".

In the INDENICA context the definition from ITIL seems relevant[2]: *"A Service is a means of delivering value to Customers by facilitating Outcomes Customers want to achieve without the ownership of specific Costs and Risks"*. The ITIL also says that: *"From the customer's point of view the value of a service consists of two basic elements:*

- *Utility is the functionality offered by a service to suit a specific need. Utility is also frequently described as 'what a product or service does'. In addition to functionality it can also mean the removal of constraints for the business. Utility increases the performance of the enterprise.*
- *The second element is Warranty, the commitment or warranty that a product or a service matches the agreed requirements concerning availability, capacity, continuity and security. The service warranty reduces the fluctuations in the service delivery."*

2.2 Service Oriented Computing

Since software development is a complex, time consuming, and expensive process, vendors as well as customers can benefit from reusable software elements. To achieve sustainable reusability, on a business-internal as well as on a business-to-business (B2B) level, a high degree of interoperability and integration is required. The concept of Service-Oriented Computing [1] (SOC) utilizes *services* as the fundamental elements for application development: services are "self-describing, platform-agnostic computational elements that support rapid, low-cost composition of distributed applications" [2]. Among the core characteristics of services are *loose coupling* (services are not hard-wired but constitute self-contained units that can be dynamically bound), *autonomy* and *abstraction* (services have control over the internal implementation logic and only provide an abstracted service interface to the outside world), *service contract* (services describe themselves and adhere to certain interface and protocol agreements), *composability* (services can be composed to create higher-level functionality) and *discoverability* (services can be found via suitable discovery mechanisms) [3]. Today, the most often used technology for imple-

[1] http://www.nexof-ra.eu/?q=node/187

[2] http://www.itil.org/en/glossar/glossarkomplett.php?filter=S

menting SOA are Web services [4]. In a nutshell, a Web service is a software application that provides a programmatic interface (expressed using the Web Services Description Language, WSDL), uses XML-based messaging protocols (Simple Object Access protocol, SOAP) and is subject to different policies that refer to domain-specific capabilities, requirements, and general characteristics.

A number of technologies, platforms and infrastructures for development and deployment of services have been proposed in recent years. The OSGi framework (formerly Open Services Gateway Initiative) [5] defines a service platform and module system for the Java programming language. It includes a Reference Architecture for service platforms, including a detailed description of involved entities and specification of services of various types. The notion of services in OSGi is very general, ranging from logging services and user administration services to IO connector services or XML parser services.

Various efforts have been made to develop service platforms for devices with resource limitations such as mobile devices [6], smart phones or automobiles [7]. Service platforms on such devices often provide contextual information to allow the implementation of context-aware services, e.g., energy-aware or location-aware services [8]. The Devices Profile for Web Services (DPWS) [9] is an effort to support the Web services protocol stack on resource-constrained devices. It includes constraints, mapping rules and lightweight alternatives for Web service messaging, discovery, description, and eventing. The SIRENA project presents a service infrastructure tailored to the requirements of real-time embedded networked devices [10].

Service platforms and infrastructures often build on a service registry, which stores endpoint information, interface definitions, and other metadata of services. The use of a service registry allows the service platform to dynamically look up, select, and bind to candidate services at runtime. In the area of Web service registries, a number of approaches and standards exist. UDDI [11], which was originally proposed as a core Web service standard, models characteristics of services (in the form of businessService, bindingTemplate, and tModel) as well as identifies service providers (businessEntity contains metadata about a publisher and publisherAssertion describes relations between parties). UDDI had very limited success and was never fully adopted by the industry. This claim is supported by the fact that public UDDI registries of Microsoft, IBM and SAP were shut down in 2005. The set of specifications collectively described as ebXML (Electronic Business using XML) [12] enables enterprises to conduct electronic business over the Internet. Amongst other concepts, ebXML defines a Registry Information Model and a Registry Services standard. Similar to UDDI, the ebXML data model is rather unstructured, reducing the service description to a collection of links to its technical specification, such as the WSDL document. A comprehensive service registry and runtime environment is VRESCo [13] (Vienna Runtime Enviroment for Service-oriented Computing). The VRESCo registry distinguishes between the metadata model, the service model and the QoS (Quality of Service) model. IBM's WebSphere Service Registry and Repository (WSRR) [14] uses a more structured information model, with the ability to automatically generate model entities (called logical derivations) from physical documents of well-known formats such as WSDL, XSD or WS-Policy.

2.3 Software-, Platform-, and Infrastructure as a Service

A cloud [15] is often used to identify the different layers (users) of the SOA paradigm. Services are used at application level (SaaS), platform level (PaaS), and Infrastructure level (IaaS). More precisely:

Infrastructure as a Service: "This is the delivery of hardware (server, storage and network), and associated software (operating systems virtualization technology, file system), as a service. It is an evolution of traditional hosting that does not require any long-term commitment and allows users to provision resources on demand. The IaaS provider does very little management other than keep the data center operational and users must deploy and manage the software services themselves just the way they would in their own data center".

Platform as a Service: "This is the idea that someone can provide the hardware (as in IaaS) plus a certain amount of application software - such as integration into a common set of programming functions or databases as a foundation upon which you can build your application. Platform as a Service (PaaS) is an application development and deployment platform delivered as a service to developers over the Web. It facilitates development and deployment of applications without the cost and complexity of buying and managing the underlying infrastructure, providing all of the facilities required to support the complete life cycle of building and delivering web applications and services available from the Internet. This platform consists of infrastructure software, and typically includes a database, middleware and development tools. A virtualized and clustered grid computing architecture is often the basis for this infrastructure software. Some PaaS offerings have a specific programming language or API. For example, Google AppEngine is a PaaS offering where developers write in Python or Java. EngineYard is Ruby on Rails. Sometimes PaaS providers have proprietary languages"

Software as a Service: "This is the idea that someone can offer you a hosted set of software (running on a platform and infrastructure) that you do not own but pay for some element of utilization - by the user, or some other kind of consumption basis. Here you do not have to do any development or programming, but you may need to come in and configure the software. You do not have to purchase anything. You just pay for what you use. A SaaS provider typically hosts and manages a given application in their own data center and makes it available to multiple tenants and users over the Web. Some SaaS providers run on another cloud provider's PaaS or IaaS service offerings"

2.3.1 Some commercial solutions

This overview has been prepared in conjunction with the 4Caast project[3] based on [16]. While [16] evaluates different solutions with different abstraction layers we restrict ourselves to solutions that can be considered a platform. This means the solution has to offer the capability to build a customer-tailored application on top of it.

The following criteria have been considered when evaluating the solutions:

[3] http://4caast.morfeo-project.org

- Domain: the (business) domain in which the solution is offered
- Type: whether the solution can be related to SaaS, PaaS or IaaS
- Target audience: the audience that is the intended customer. As we focus on developers in the INDENICA project, the audience should match. However, often the border between a developer and a typical end-user is blurry.
- Integration: the way of integration of the solution into an own application. Ideally the solution should provide a complete framework for application building.
- Extensibility & Variability: the mechanisms that are used to extend the core functionality offered by the solution. This includes provided variation mechanisms that allow changing certain aspects of the solution. (Configuration, Extension Points, APIs (REST, WS, Native), Language)
- Business Model: the business model that is behind the solution.

Amazon Web Services

Amazon Web Service is only a brand for various offerings that can be assigned to the area of IaaS. These offerings include S3 (storage service), EC2 (virtual machines), EBS (storage volumes) and VPC (virtual network for EC2 instances).

Domain	No particular, technical only.
Type	IaaS
Target Audience	Developers
Integration	EC2 provides a whole operating system in a virtual machine, for this reason a developer is completely free to choose the integration. The actual mechanisms depend on the applications installed inside the VM. EC2 itself can be controlled via SOAP-WS, bindings for various languages also exist. S3 provides access to files via so-called buckets. Access is granted over HTTP. EBS and VPC are used in the backend only; they are not intended for extension.
Extensibility & Variability	As a customer has full control over an EC2 instance he/she is completely free extending the functionality. The other offerings are not intended for extensions.
Business Model	Pay-per-use, based on memory and bandwidth consumption.
URL (if available)	http://aws.amazon.com

Gigaspaces

Gigaspaces advertises its product XAP Elastic Application Platform as “Industry’s only virtual application platform”. An Elastic Application Container is flanked by a de-

ployment infrastructure that serves as a bridge to existing cloud infrastructures and an interfacing layer that is use as the basis for application building.

Domain	No particular, technical only.
Type	PaaS
Target Audience	Developers
Integration	XAP provides various API including core read/write, key/value storage, JDBC, POJO domain model and monitoring. Actual application can be developed as J2EE, .NET or Spring applications and can therefore use the full freedom these frameworks offer.
Extensibility & Variability	The platform itself does not seem to be intended for extensions.
Business Model	Pay-per-use.
URL (if available)	http://www.gigaspaces.com/

Google AppEngine

AppEngine is Google's PaaS offering. A user is able to develop web application in either Python of Java and deploy the application into the "AppEngine cloud".

The services that are offered (and that can be used programmatically) by AppEngine are of rather basic nature. This includes (among others) fetching internet content, e-mail, caching, image manipulation, data storage and user authentication.

Domain	No particular, technical only.
Type	PaaS
Target Audience	Developers
Integration	The customer develops a web application either in Python or Java. Google provides appropriate APIs to use the functionality of the platform. The application itself may offer SOAP or REST services to be used externally.
Extensibility & Variability	A customer is free to structure his application based on the API the solution offers. The platform itself is not intended for extensions.
Business Model	Pay-per-use. The fee is not based on resource consumption, but on the number of deployed applications itself.
URL (if available)	http://code.google.com/appengine

Heroku

Heroku is a multi-tenant platform targeting Ruby developers.

Domain	No particular. A focus on a certain domain can be established by using predefined extension modules.
Type	PaaS
Target Audience	Developers
Integration	The development is performed in Ruby. All Ruby APIs can be used. For this reason the development can be done locally, the finished application is deployed into the Heroku cloud afterwards.
Extensibility & Variability	Heroku offers a number of extensions, which are mainly connectors to third-party services. These extensions can be booked by the customer and are either free or need to be paid. Other extension mechanisms are not documented.
Business Model	Pay-per-use, based on number of processes and number of active users.
URL (if available)	http://www.heroku.com

Microsoft Azure

Azure is Microsoft's offering for Cloud services. It consists mainly of three parts: Windows Azure, SQL Azure and AppFabric. The first one is the Windows operating systems itself, that is offered on-demand, the second provides SQL storage capabilities and the latter is a set of .NET-based components that can be used to build cloud-aware applications.

Domain	No particular, technical only.
Type	Both IaaS and PaaS
Target Audience	Developers
Integration	For the Windows Azure Offering the customer is free to use and develop his/her own solution. Therefore the integration mechanisms depend on the actual applications installed inside an instance. The SQL Azure offering can be integrated via a REST-interface. For AppFabric Microsoft provides APIs for all .NET languages.
Extensibility & Variability	The platform itself is not intended to be extended. The application creation is bound to the capabilities the API offers.

Business Model	Microsoft allows pay-per-use depending on computational resources and also a subscription model.
URL (if available)	http://www.microsoft.com/windowsazure

NetSuite

NetSuite is both a SaaS and a PaaS solution. The company offers three on-demand applications for financial management, customer relationship management and eCommerce. In addition it is possible to use the underlying platform as a PaaS solution (called SuiteCloud resp. NetSuite Business Operating System (NS-BOS)).

Domain	Business applications
Type	SaaS and PaaS
Target Audience	Developers and end-users
Integration	NS-BOS offers a number of APIs which can be used from a JavaScript script. In addition WS-APIs exist. The APIs are rather high level; access to lower level functionality of the platform is not supported.
Extensibility & Variability	Extensions can be developed by using the provided JavaScript-API, further (low-level) mechanisms (as indicated above) are not supported.
Business Model	Not documented on the website.
URL (if available)	http://www.netsuite.com

OrangeScape

OrangeScape is solution that allows the development of business applications using a visual style modelling interface. It is supported by a development environment called OrangeScape Studio that runs inside a web browser. The actual applications can be deployed to other PaaS solutions including Google AppEngine, Microsoft Azure and Amazon EC2 or to conventional environments like J2EE and .NET in an on-premise fashion.

Domain	Business applications.
Type	PaaS
Target Audience	Developers, advanced end-users
Integration	OrangeScape offers five main services: a component service for defining business concepts, a workflow service, persistence, web access and an AJAX-based presentation layer. The web access service exposes functionality via REST and can be used for integration purposes.
Extensibility & Variability	The OrangeScape Studio (the development environ-

	ment) allows the design of data models, forms, processes and actions in a visual way. The developer is bound to the functionality of the visual designer component, further extensions are not supported.
Business Model	Pay-per-seat, based on the number of developers accessing the web-based development environment.
URL (if available)	http://www.orangescape.com

Salesforce

Salesforce provides the following offerings: a SaaS solution in the area of Customer Relationship Management (CRM) ("Salesforce CRM") and a development platform called force.com. The latter consists of three parts, namely a collaboration platform, a development platform and a cloud infrastructure.

Domain	Customer Relationship Management
Type	Saas and PaaS
Target Audience	End-users, developers.
Integration	Integration can be achieved via WS. In addition there are public APIs. Salesforce delivers its own programming language called Apex.
Extensibility & Variability	Not possible for Salesforce CRM. The force.com platform can be used for creating dedicated applications, which are offered on a marketplace. These application must be developed in Apex and make use of the API that is offered by the force platform.
Business Model	Subscription model.
URL (if available)	http://www.salesforce.com and http://www.force.com

WorkXPress

WorkXPress is a platform that enables customers to build (business) applications based on a 5th generation language (5GL). The actual "programming" is done entirely visually.

Domain	Business applications.
Type	PaaS
Target Audience	(Advanced) End-users.
Integration	WorkXPress offers a SOAP-based API. No other integration possibilities are documented. The applications can be deployed into commercially available clouds like Amazon EC2.
Extensibility & Variability	The user is bound to the capabilities of the visual edi-

	tors the vendor ships. No other extensions are foreseen (resp. documented).
Business Model	Pay-per-seat and pay-per-use combination based on the number of users and the costs of the cloud solution the application is deployed to.
URL (if available)	http://www.workxpress.com

2.4 Service Component Architecture (SCA)

The Service Component Architecture, a standard of the Open SOA Consortium [17], integrates the service-oriented paradigm with component-based development (for an overview see [18]). In SCA (Figure 1) components are the building units of modules and composite applications that communicate with each other via services. Components are hosted in SCA containers in such a way that the component developer is shielded from the underlying container technology such as EJB, OSGi, Spring or WCF. For data-source-independent persistence the SDO standard has been provided. There are efforts to create a SCA .NET binding.

Figure 1: Composing components to modules in SCA.

SCA represents a potential technology for INDENICA as it is standardized, supported by many vendors and allows a higher level of integration than services that might be useful when marrying PLE and SOA.

2.5 Software Product Lines and variability management

Software Product Lines have gained increasing attention as an approach to systematically develop related software products as a set, which form a so-called product line. This approach has moved the focus from independently developing single systems to systematically engineering a landscape of similar products. In [19] the authors define a software product line (SPL) as a *"set of software-intensive systems sharing a common, managed set of features […] that are developed from a common set of core assets in a prescribed way"*. This section gives a brief introduction to the main ideas and the general definitions used in this area.

2.5.1 Business orientation

The basic motivation to employ product line engineering is an economic one: Profound planning and management of reuse should lead to significantly reduced costs and time-to-market compared to independently developed systems.

Product line engineering focuses on establishing a sustainable family of products instead of only developing a single project successfully. This demands a more holistic perspective on the business strategy. Its major goal is not only a timely and cost-effective delivery of the current product, but must be extended to a strategic view on the business area that should be covered by a complete range of products, which form the product line.

The product line approach implies a higher initial effort for building reusable assets, restructuring development processes etc. After an initial investment, which is needed to set up a product line, significant savings of costs and time usually occur as early as after three products [20]:

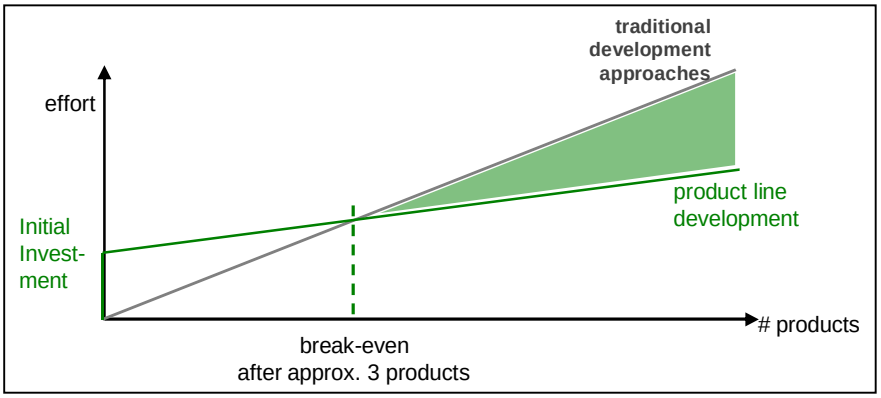

Figure 2: Economics of software product lines [19].

2.5.2 Management of variability

The abovementioned definition of product line focuses on a “common, managed set of features”. Thus a grounded, systematic analysis and management of *commonalities* and differences between several products in a product line is crucial to their successful engineering. This leads to the discipline of *variability management*. It further distinguishes the aforementioned differences of products between variabilities and product-specific additions as illustrated in Figure 3:

- Variabilities are characteristics that vary between products in the sense that some (more than one) products share them, whereas other products do not feature them at all. In Figure 2 these appear as the triangular area at the intersection of the rectangular-shaped products.
- On the other hand a product-specific characteristic is unique to a single product.
- Commonalities define the core shared of all products in the product line, highlighted as the pentagon-shape in the center. They are developed once for usage in all products of the product line.

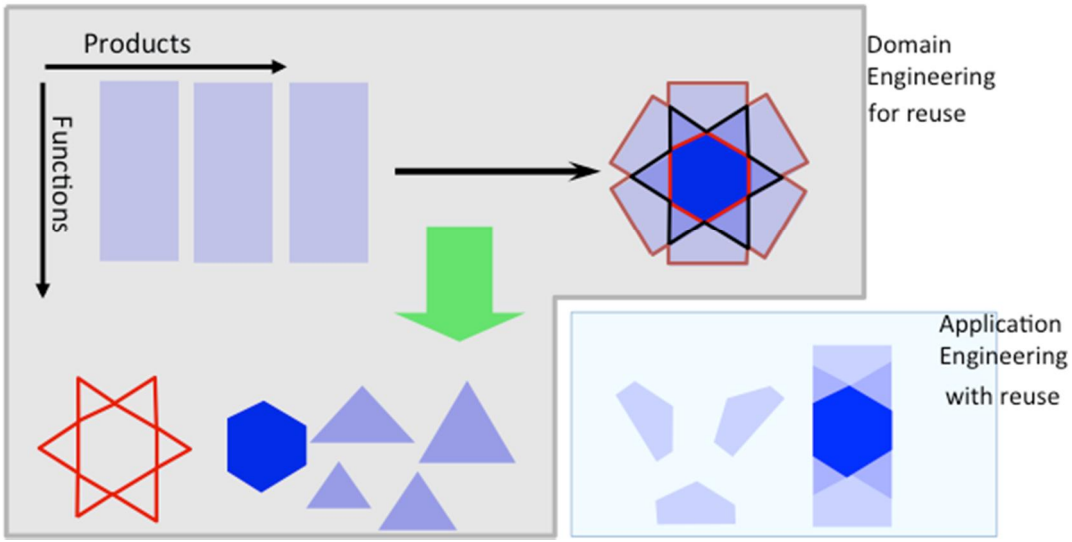

Figure 3: Commonalities, variabilities and product specifics in domain engineering

A successful and cost-effective product line requires thorough planning and engineering from the beginning. At first it is important to understand the range of relevant variation. This is also called scoping [21].

Then it is essential to clearly define which parts belong to the core of all products, which are shared by some, and which are specific to one product. This information of variability is captured in the variability model. It includes information on the dependencies among variations and constraints, under which circumstances certain variations can occur. Research has developed a number of approaches on how to model variation. These are usually embedded in variability management techniques as described below.

The common basis of the products is developed in *domain engineering*: From the beginning all assets are not developed with a single product in mind, but rather designed *for reuse*. The domain, defined by the products that shall be developed, is analyzed for commonalities and variabilities. In *application engineering* they are combined with product-specific parts to form the final instances of the product line.

2.5.3 Variability management techniques

Many surveys have been developed to give a comprehensive view on existing variability management techniques [22,23,24]. The most common approaches are feature-based modelling [25,26] and decision-based modelling [27]. Below we present a short overview of these approaches. We do also briefly discuss additional variability management techniques that cannot be assigned to these categories. Finally, we describe techniques, which address large-scale Software Product Lines.

Schobbens et al. focused mainly on feature-based modelling approaches [28,29,30]. They compared Feature Oriented Domain Analysis (FODA) [25] with its extensions, such as the Feature-Oriented Reuse Method (FORM) [31], FeatuRSEB [32] and feature diagrams with multiplicities [33]. Based on this comparison Schobbens et al. developed a formal semantics to avoid redundancies and inconsistencies. Van den

Broek et al. deal with issues of merging feature diagrams together [34]. Additionally, they also provide a formal definition of such features.

Decision Modelling is a second family of approaches to model variability. It was initially described in [35]. An important input to variability modelling in the INDENICA project will be the approach by Schmid and John [36]. In this technique, decision variables are defined, which are referenced at the specific variation points using decision evaluation primitives. This approach is explicitly open to be used with a number of different modelling technologies. This makes it well suited for the different types of technologies relevant to this project. A comparison of Decision Modelling techniques is given in [316].

Beside the two families of approaches, feature-based modelling and decision modelling, there exist a lot of other techniques to handle variability that fall not under the aforementioned categories, e. g., modelling variability with UML diagrams [37,38]. A problem that can occur is the resulting complexity of handling the variability, because variability information is spread across different models. Pohl et al. introduced the orthogonal variability model, which provides a centralized view of variability information, to handle this issue. Within this approach, it is possible to link this variability information with other artefacts, like requirements or UML diagrams [39].

Much current work in variability management deals with handling of complex and large-scaled software product lines. The aim of approaches like multi-level staged configuration [306], multi-dimensional variability modelling [307], model fragments [317], hierarchal variability modelling [308], and Product Populations [309] or Multi Software Product Lines [319, 320, 321] is to avoid having all variability information in one central complex variability model.

Multi-level staged configuration provides separate feature models for every stage of configuration where different persons are responsible for the configuration choices. Multi-dimensional variability modelling offers separate variability models for the different dimensions of variability (e.g. feature dimension and architectural dimension); also the previously mentioned orthogonal variability model belongs to this category. Model fragments allow to decompose a complex variability model into small pieces. For each of these fragments, different stakeholders can be assigned, which are able to edit these fragments. With hierarchical variability modelling the variability specification is distributed over a hierarchical set of components decomposing a complex system. Product Populations or Multi Software Product Lines go one step further and facilitate to form products or even new product lines out of more than one product line infrastructure. In this approach, each product line consists of a variability model and accompanying assets. Several of these are than composed to form the final products. A comparison of model fragments and multi software product lines with other useful techniques for complex product lines can be found in [318].

Regardless of what variability management technique is used, modelling of further constraints must be supported to manage interdependencies among different aspects. These constraints can be used to validate product configurations. Along these lines several works originated that deal with the representation of those models, expressiveness and the (efficient) validation of those models [40,41].

2.6 Model-Driven Development

Model-driven development (MDD) is an emerging software development methodology aiming at enhancing development speed and software quality [42,43]. In MDD, models are first-class artefacts that can be used not only for documentation and communication solely, but also for many other purposes, such as reasoning about business or solution domains, analyzing the architecture of the solution, generating code, and so on, in the software life cycle [42,43,44].

[45] defined a model as a coherent set of formal elements describing something (e.g., a system, bank, phone or a train) built for some purpose that is amenable to a particular form of analysis, such as communication of ideas between people and machines, completeness checking, race condition analysis, test case generation, viability in terms of indicators such as cost and estimation, standards, and transformation into an implementation.

[46] presented a formal framework for MDD approaches, in which, the definitions of models and the system and their relationships are given as follows: System is a delimited part of the world considered as a set of elements in interaction. Model is a representation of a given system, satisfying the substitutability principle. A model is said to be a representation of a system for a given set of questions if, for each question of this set, the model will provide exactly the same answer that the system would have provided in answering the same question.

In this light, the OMG's MDA specification [47] can be seen as one specific MDD approach that is different from the MDD approach in general. The MDA primarily focuses on interoperability, platform independence, and is merely based on, as well as often limited to, OMG specifications such as MOF [48], UML [49], OCL [50], etc. MDD however is not bound to specific standards or technologies and advocates the idea of using customized, tailored domain-specific languages (DSL) to capture precise representations of structure, function or behaviour of systems or software in a particular domain [43].

A domain under consideration may be divided into smaller sub-domains. Domain-specific languages (DSLs) are usually used for modelling domain concepts and knowledge in MDD. DSLs are small, sometimes declarative languages that can offer powerful expressiveness through appropriate notations and abstractions of a particular problem domain [50,51,52]. The most important characteristics of DSLs, with respect to general-purpose languages, are the compactness and expressiveness in a certain domain, such that domain experts themselves can understand, analyze, validate, modify, and even develop DSLs [51,52,53].

A DSL has one or many concrete syntaxes, which are either textual or graphical. A DSL's concrete syntax can be used to define formal models. This concrete syntax is based on a language model (abstract syntax) [54] that specifies the structure and static semantics of the DSL's concrete syntax. A DSL's abstract syntax is embodied in a meta-model. Thus, DSLs are sometime mentioned as modelling languages. The model, i.e. the DSL's abstract syntax, has to conform to a meta-model that specifies structure and the semantics of that model [43].

Model transformation plays a very important role in which another model can be created from a source model according to some predefined mapping rules [55,42,43]. For instance, a platform-independent model (PIM) is mapped into a platform-specific model (PSM), or code is generated from a PSM [55,43]. The mappings between models, i.e., PIM to PIM or PIM to PSM, are model-to-model transformations, while the generations of code from PSMs are model-to-code transformations (or so-called, code generation) [55,43]. As such, model transformations establish relationships between models at the same or different levels of abstraction as well as between models and generated code. Therefore, they become very important factors in MDD for enhancing development automation and bridging abstraction levels. The results of code generation are usually schematic recurring code fragments that form the basic skeleton of the systems or software under development. The rest must be filled by non-generated code (or so-called, individual code or handwritten code) that is manually implemented [43].

The role of model-driven development in the context of product line engineering is an emergent topic [305]. Both PLE and MDD have shortcomings. As Kim et al. state in [303], instructions and artifact templates in PLE are not precisely defined, resulting in a still conceptual model as final deliverable. MDD however does not model resuable assets to be used in product variants. [303] grasp the instantiation of a product variant based on a feature model as usage of model-driven development techniques. Furthermore they define a process for model-driven PLE that allows compensating of these shortcommings and benefit from their synergy.

Schaefer [311] presents a model-driven approach to manage variability of software product lines. For a core model, representing a valid product variant of the product line, several so-called Δ-models allow the derivation of other product variants. Those Δ-models describe transformation steps to construct the variant out of the core model. This approach is orthogonal to the overall refinement process as the core and the Δ-models are refined independently for every refinement step.

In [302] domain-specific languages are used to fill the gap between the feature model and the programming language. DSLs allow a better description of complex feature specifications by still being in the problem space instead of using general purpose programming languages from the solution space. By combining feature-modelbased PLE and DSLs, they achieve benefits for managing variability and traceability.

Tracebility is important in PLE as it provides a way to follow requirements forward and backward in the product lifecycle. This allows e.g. to analyse the impact of changing requirements in a product line environment, or the originating features of a concrete variant. The Ample Project [304] provides a combination of MDD techniques in the course of PLE to examine tracebility in SPLs. Furthermore, MDD is used in all stages of the SPL to express variability allowing requirements refinement to architecture.

Stahl and Völter state a number of advantages that MDD brings to software development, such as increasing productivity through automation, enhancing software quality and reusability by generating code from proven patterns and architectures, and improving manageability of complexity through appropriate abstractions [43]. In

the design space of INDENICA, different stakeholders such as platform integrators, software architects, developers, etc., are involving in the development of applications integrated functions provided by various heterogeneous service platforms at different levels of abstraction with different expertise. Thus, MDD can potentially support the stakeholders to work with the most appropriate perspective and abstraction level for his current work task. In particular, we use the MDD paradigm to realize the separation of abstraction levels in order to organize the process representations according to specific stakeholder interests; for instance, high-level representations used by business and domain experts, whilst technology-specific representations are employed by IT experts.

By combining MDD for separating abstraction levels with the notion of architectural views for separating various concerns of the design space, we aim at supporting the stakeholders in efficiently dealing with the heterogeneity and complexity of the service-platforms involving in the development of a certain service-based application. In addition, MDD transformation techniques shall be exploited to enhance the automation of producing relevant code, configurations, directives, etc. that are necessary for integrating and tailoring service-platforms. In the course of model transformations, existing formal validation and verification techniques can also be used to ensure the validity, integrity, and soundness of models being used to capture functionality and properties of service-platforms.

3 Requirements

Requirements engineering is one of the core disciplines within the product development process. As basic discipline at the very front end of the development life cycle, requirements are a critical success factor for each [software] development project [56]. Requirements [57] require both management and development.

Management [58] includes all activities about existing requirements as:

- Tracing requirements back to their origin
- Managing cross-references between requirements
- Tracing requirements forward to their implementations
- Managing requirements changes

Development describes all the activities related to getting the correct requirements for a certain product and its components [58]. Most publications differentiate among the following requirements engineering activities (Figure 4):

- Requirements Elicitation
- Requirements Analysis
- Requirements Specification
- Requirements Validation

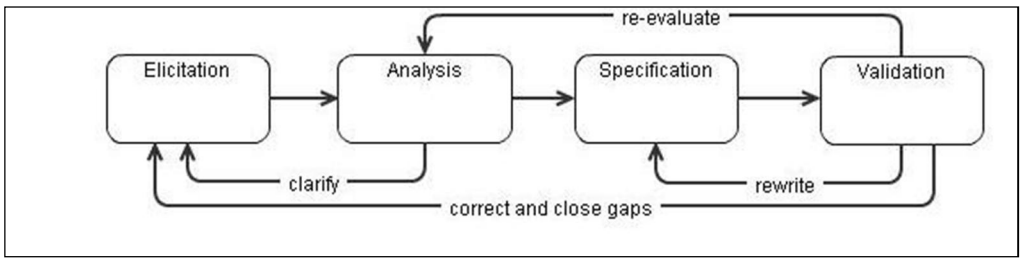

Figure 4: **requirements engineering process [57].**

Requirements elicitation

The main challenges concerning requirements elicitation are [56]:

- Identification of relevant requirement sources: Many different stakeholders may serve as sources for various requirements. As the needs of these stakeholders usually are not consistent but sometimes even contradictory, it is important to identify and prioritize these stakeholders and to take them into consideration according to their relevance for the product to be developed.
- Elicitation of existing requirements from the identified sources: By the use of appropriate elicitation methods, as interviews, workshops, focus groups etc., it is essential to reveal the real requirements from the different stakeholders sometimes hidden behind diverse unstructured statements.
- Development of new and innovative requirements: Beyond the requirements coming from the identified requirement sources, it is a crucial task to develop

proactively ideas which can finally lead to new up to now unknown requirements.

Requirements analysis

The main challenges concerning requirements analysis are [56,59]:

- Eliminate unnecessary requirements: Identify unnecessary requirements by clearly defining scope and boundaries of a system and eliminate them.
- Structure requirements: Find the right way to structure requirements thematically and arrange them hierarchically.
- Add missing requirements: Complete the set of requirements by identifying gaps within the hierarchical structure and filling these gaps with appropriate requirements
- Discover requirements overlaps and conflicts: Each stakeholder has his own view to the product to be developed. In this process step requirements duplicates have to be eliminated, and conflicts have to be revealed.
- Analyze the cause for each conflict: Conflicts among requirements may arise from diverse conflict types: data conflict (e.g. lack of information), interest conflict (different interests or goals of stakeholders), value conflict (different criteria in evaluating an issue). It is crucial to make the type of conflict transparent as base for negotiating the requirements in the next step
- Resolve the conflicts: Find the right balance to resolve the requirements conflicts by negotiating, proposing new solutions and final deciding.
- Document the resolution and their rationale: In order to prevent discussions in the future, it is essential to document the resolution of the conflict together with its rationale.
- Prioritize requirements: Find the right prioritization among the analyzed requirements by means of pair wise comparison or other appropriate methods.

Requirements specification

The main challenges concerning requirements documentation are [56,57]:

- Find the appropriate way of documenting requirements: There are different ways of documenting requirements, as: natural-language documentation, documentation by graphical models, documentation by formal specifications. Depending on the affected stakeholder groups, identify the most appropriate way of documenting the requirements.
- Comply with required quality criteria for requirements documentation: Select appropriate set of quality requirements for documentation in terms of guidelines. Use of existing reference templates may serve for improving the quality of requirements documentation.
- Define appropriate set of attributes for requirements: In order to structure information about requirements and keep them up to date along the lifecycle of the requirement, it is essential to assign and maintain carefully well-defined attributes for each requirement.

Requirements validation

The main challenges concerning requirements validation are [56]:

- Check the requirements artefacts: Check whether the requirements artefacts meet the defined quality criteria (unambiguous, complete, non-contradictory)
- Check the context consideration: Check for missing or incorrect context information
- Check adherence to process definition: Check for deviations from agreed development process

3.1 Requirements engineering for Product Lines

As in [108] a product line is a group of products developed on basis of common, organized, reusable artefacts of a platform. It covers the market needs of a specific domain or business. This implies that the products must have a set of common or similar features big enough, so that the development as a product line is more efficient as developing single products.

Main goal of a product line is to provide customized products at reasonable costs. With the platform as basis for the product line, strategic reuse is on of the main principles. A well-planned platform can help to reduce the development costs, enhance the product quality and reduce time to market for a single product. Product Line Engineering (PLE) helps to get the right setup, processes and methods for developing product lines

The main characteristics for Product Line Engineering (PLE) consist in [39] [108]:

- The existence of two different development processes:
    - Domain Engineering: The process of software product line engineering in which the commonality and the variability of the product line are defined and the reusable platform is established.
    - Application Engineering: The process of software product line engineering in which the applications of the product line are built by reusing domain artefacts and exploiting the product line variability
- Variability as a core concept for PLE
    - It defines the parts of the product line which can be tailored to specific customer needs by selection. It also defines how much members of a product line differ.
    - It is a prerequisite for the systematic construction of the artefacts in domain engineering and their reuse in application engineering.
    - For more details on variability modelling see chapter 2.5.
- A platform for the product line:
    - It contains common reusable development artefacts created by domain engineering like requirements, architecture, variability models, components and tests.
    - It is built by domain engineering with the necessary variability in its artefacts.
    - Systematic and consistent reuse is supported by traceability links between the artefacts.
    - It is used by application engineering for building the specific products by binding the variability. As a result the platform is customized to the specific product.

- A reference architecture
    - o which captures the high–level design of the product line.
    - o Which implements architecture relevant product line requirements.
    - o Which contains variation points for realizing the different customer specific products.

Figure 5 illustrates the different process steps and highlights where requirements engineering is located.

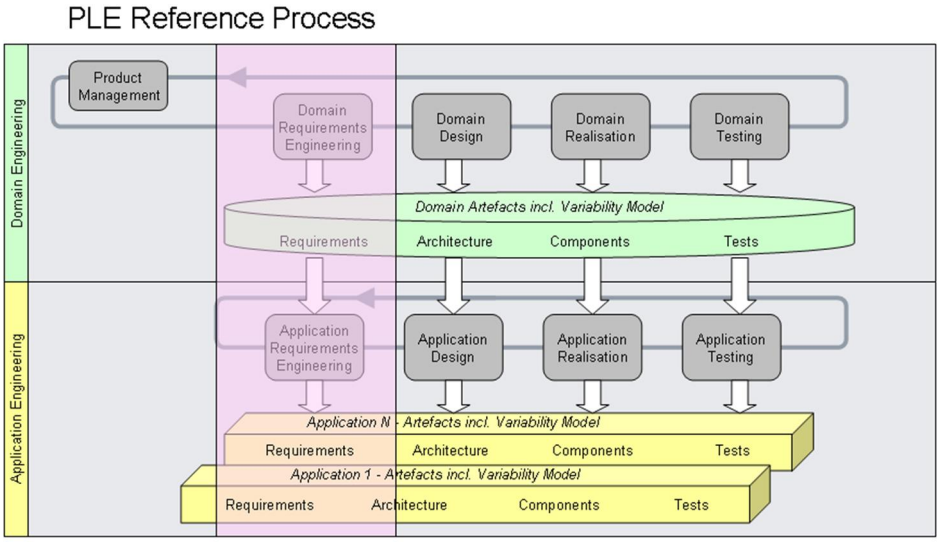

Figure 5: Product Line Engineering Process [39].

Impact of PLE on Requirements Engineering (RE)

The splitting into two development processes has impact on the requirements engineering. This has to be done in both parts and has to be synchronized so that domain artefacts and products fit together.

Domain Requirements Engineering:

In Domain engineering, the variability of the product line is established. In this case RE has special activities for building the variability into the domain artefacts:

- Defining common requirements by performing a commonality analysis
- Defining variable requirements by performing a variability analysis
- Defining variation points and variants with the help of variability modelling:

In addition domain Requirements engineering has to provide consistent requirements artefacts

Application Requirements Engineering:

In Application Engineering the application is built on basis of the common platform by binding variability. Application Requirements Engineering therefore has to document the application requirements artefacts by reusing as much as possible from the existing domain requirements artefacts. The reuse of the domain requirements artefacts is correlated to the reuse of the domain artefacts.

The special activities are

- Check the feasibility, if the stakeholder requirements can be satisfied by the application requirements artefacts
- Check the reuse possibilities by mapping stakeholder requirements to the domain requirements artefacts.
- Check for requirements deltas by mapping application requirements artefacts to domain requirements artefacts
- Communicating the commonality and external variability of the product line to increase the reuse
- Evaluating the deltas between domain and application requirements for estimating the realization effort.
- Documentation of the application requirements as basis for later development phases

3.2 Goal-based approaches

Goals have long been recognized to be essential components involved in the requirements engineering (RE) process. Requirements must specify why a system is needed, what system features will serve and how the system must be constructed [60]. In this context goals specify the objectives the system under consideration should achieve and the properties that have to be ensured.

Goals can have different levels of abstraction: high-level strategic goals can be refined into alternative (OR-refinement) or mandatory (AND- refinement) subgoals, representing low level technical concerns. Traditionally, goals cover two types of (conventional) requirements: functional requirements associated with the services to be provided, and non-functional requirements associated with quality of service – such as safety, security, accuracy, performance, and so forth. Another important distinction is between soft goals [61], whose satisfaction cannot be established in a clear-cut sense, and hard goals [62] whose satisfaction can be assessed through verification techniques. Soft goals are especially useful for comparing alternative goal refinements and choosing one that contributes the best to them.

Goals are to be fulfilled by the system-to-be, which comprises the software and its environment and is made of active components, also known as agents, such as humans, devices and software. A goal may, in general, require the cooperation of a hybrid combination of multiple agents to achieve it. A goal under the responsibility of a single agent in the software becomes a requirement whereas a goal under the responsibility of a single agent in the environment becomes a domain assumption. Unlike requirements, assumptions cannot be enforced by the software-to-be, they will hopefully be satisfied thanks to the system domain.

Goals are considered a fundamental instrument to model requirements for several reasons [63]:

- They provide a criterion to establish the completeness and pertinence of requirements[4];

[4] The specification of requirements is complete, with respect to a set of goals, if all goals can be proved to be achieved from the specification and the properties known about the domain considered. A requirement is pertinent, with respect to a set of goals in the domain considered, if its specification is used in the proof of at least one goal.

- Alternative goal refinements offer the possibility to explore a wider set of choices. Goals also provide the roots for detecting conflicts among requirements and for resolving them. Managing conflicts among multiple viewpoints is another major RE concern.
- Goals implicitly support requirements evolution, since a requirement represents one particular way of achieving a specific goal; then, the requirement is therefore more likely to evolve, towards another way of achieving the same goal, than the goal itself.

An alternative way to model requirements is the adoption of scenarios [64]. A scenario is an example grounded in the real world experience that illustrates a typical sequence of interactions among system components to meet an objective that is left implicit. The main drawback of scenarios is that they are too partial, since they do not cover the whole system behaviour under all possible circumstances. Instead, goals are global, as they specify all the intended properties of the system. Scenarios and goals are complementary techniques since scenarios can be adopted to validate requirements specification as test cases or counter-examples. Among goal-oriented methodologies we analyze i* [65], TROPOS [66], and KAOS [67].

i* is tailored to the early phases of requirements engineering and focuses on the stakeholders and their needs. These are represented in terms of agents having intentional properties such as goals, beliefs, abilities and commitments. It adopts a Strategic Dependency (SD) model to explicitly represent dependencies among actors that may exist for goals to be achieved, tasks to be performed, and resources to be allocated. The Strategic Rationale (SR) model, in turn, focuses on a single actor and includes its interests and concerns, and how they might be addressed by various configurations of systems and environments. Because of its focus on the stakeholders, this methodology is oriented towards agents, their dependencies and goals. As a result, it does not address goal traceability onto late requirements/specifications and the system architecture. Furthermore i* does not provide an explicit representation of the environment.

Instead, TROPOS is an agent-oriented software engineering (AOSE) methodology that covers the whole software development process. It reuses the concepts provided by i* and provides a way to express goals through a temporal specification language [68]. TROPOS describes the system-to-be as one actor that has a number of dependencies with other actors of the organization. These dependencies drive the definition of functional and non-functional requirements. TROPOS links requirements to the system architecture, which is defined in terms of subsystems (derived from the actors) and interconnected through data and control dependencies (supporting the functional and non-functional requirements mentioned above). The behaviour of each architectural component can be executed on an agent platform.

Conversely, KAOS does not take into account the dependencies among actors, but adopts a top-down centralized point of view. KAOS incrementally elaborates four complementary sub-models: the goal model, the object model, the agent responsibility model and the operation model. Each goal can be specified formally through a temporal language. From the definition of each goal, the main objects (entities and events) populating the system-to-be and the environment are detected, together

with their relationships and attributes. The agents are also identified and are associated with the goals they are responsible for. Finally, operations represent the tasks that must be performed for the achievement of a (leaf) goal and are specified in terms of pre- and post-conditions.

Although all the methodologies presented above are sound, we are convinced that KAOS is more suitable for the design of adaptable and evolvable solutions. TROPOS and i* are more focused on stakeholder interactions and have been mainly adopted for agent-oriented platforms. On the other hand, KAOS adopts a centralized approach, which is similar to the point of view of the process provider, and focuses on the operations and objects that must characterize the environment and the software of the system-to-be. The operation, object, and agent responsibility models provide a clear way to define a link among the elements of the goal model (operations, objects, agents). Furthermore the formal definition of goals allows us to reason about why/how/when process activities must be performed at runtime and how their supervision can take place.

3.3 Requirements for adaptive systems

In general, goal models provide several nice features to support self-adaptive systems. They embed variability by expressing a large set of possible behaviours. Possible variability points [69] can be detected by alternative goals refinements (OR-refinement), different contribution of operations/plan to the satisfaction of soft goals, and the not mandatory nature of AND-refinements. A different behaviour can be selected depending on a set of conditions expressed on the context (unintentional variability) or user preferences (intentional variability) [70], since a specific alternative in the goal model better contributes to some soft-goal than other alternatives. Goals [71] allow one to reason about obstacles to their fulfilment since violation of leaf goals can be propagated to higher-level goals. Goals also facilitate the diagnosis process, since, when a change in stakeholder requirements is detected at runtime, the goal model can be used to re-evaluate the behaviour alternatives and determine if a system reconfiguration is needed.

Despite their nice features, goal models have proved to be too general to represent self-adaptation capabilities along with the conventional requirements of the system [72]. Adaptation must become a requirement “per-se” and must be elicited along with the other conventional requirements of the system at design time. As already stated by Berry et al. [73], the achievement of this objective, especially for context aware applications, depends on the success at specifying their monitoring and switching behaviour. According to this view, we retain that the goal model must represent why/when/where/how adaptation must be performed: this is equivalent to describing respectively the adaptation objectives, the monitoring, diagnosis and the adaptation action a system must support.

Some preliminary solutions [71,74] have already tried to achieve this objective by adopting goal models to elicit the adaptive behaviour of the system. Lapouchnian et al. [71] exploit alternative paths in the goal model to derive a set of possible system behaviours. This way, when a requirement is violated, these alternatives are ready to be selected to perform adaptation. Goldsby et al. [74] use four i* goal models to rep-

resent the different dimensions that characterize a DAS (dynamically adaptive system). First, the non-adaptive behaviour of the system (business logic) is modelled. Second, the adaptation strategies are designed to handle environmental changes. At this level, an adaptation scenario is created to represent the requirements of the monitoring mechanisms, decision-making mechanisms and adaptation mechanisms necessary to accomplish an adaptation, conceived as a transition from a steady state system to another one. Then, at the lower levels, the mechanisms needed to perform adaptation are represented. In particular, the third level identifies the capabilities the adaptation infrastructure must expose to support the scenario devised by the first two levels, and the fourth level depicts the various types of adaptation offered by the infrastructure.

The main problem of these approaches is that adaptation is only handled by enumerating all alternative paths at design time and there is no way to associate these alternatives to the changes that can take place in the context, the satisfaction levels of goals and the contributions to non-functional requirements of the system. Furthermore these models do not provide explicit support to unexpected adaptations. Instead, adaptation is expressed as enumerations of predefined alternative tasks.

To ensure the continuous satisfaction of requirements, one needs to adapt the specification of the system-to-be according to changes in the environment (context). This idea was originally proposed by Salifu et al. [75], and has been extensively exploited in other works [69,78,79,80]. Salifu et al. [75] provide a way to express monitoring requirements (what applications must do to detect changes in their operating environment that may violate their requirements), and switching requirements (what applications must do to restore the satisfaction of such requirements by adapting their behaviour). However the authors focus on making sure that the same set of requirements is met in every context, without considering the effects of domain variability on requirements and on the adaptive system design.

Context variability may also affect the capability of an agent to satisfy a goal. For example, Penserini et al. [76] introduce the concept of opportunity that indicates a set of context conditions (e.g., plans/soft goals contributions and environmental constraints) necessary for an agent to execute a specific plan associated with the achievement of a goal.

An important dimension in modelling the context is the location. To this aim, Dalpiaz et al. [69] extend TROPOS to model location variability. They associate each location with the behaviour alternatives (specified in the goal model) that can be performed, are impossible to execute, or are recommended. The authors also perform a set of analysis on the goal model [77]: LGS (location based goal satisfiability) that detects if a goal can be achieved in the current location instance, LPS (Location Properties Satisfiability) that diagnoses goals violations and suggests ways for solving the problem, Preference Analysis (PA) that selects the best alternative among a set of recommended/viable solutions. However context is not only limited to location properties and a more general notion of context must be provided. For example Ali et al. [78] represent the context hierarchically. The hierarchical context analysis has the potential to make the context more understandable and reusable. In fact the context is not described in terms of a monolithic block and some of its parts can be associated with

a set of variation points in the goal model or other stakeholders context specifications. Finally, another work [79] proposes to adopt ontologies to express application domain and operational context assumptions. These ontologies are linked together to enable monitoring and adaptation at runtime.

Environment variability may also affect stakeholders' goals and their refinements. Lapouchnian et al. [80] explicitly represent the effects of the context on the requirements model. In particular, the authors identify the elements of the model that depend on the context, and define contextual tags to capture the conditions that those elements require to be visible in the model.

All these works are interesting for their capability of addressing adaptation at requirements level, but they mainly target context-aware applications and adaptation.

Cheng et al. [81] consider both the environment and the satisfaction of existing goals among the factors that may trigger an adaptation. The environment is represented in terms of an UML class diagram that identifies the key physical elements of the system and their relationships (sensors, user interfaces). This is useful to identify the sources of uncertainty and to monitor the environment conditions that pose uncertainty. Threats are also identified: they are the various environmental conditions that pose uncertainty at the development time and thus may be warrant dynamic adaptation at runtime to ensure acceptable behaviour. Mitigation can be performed according to 3 possible tactics: define a further subgoal to handle the threat condition, if partial satisfaction of a goal is tolerable; add a new higher level goal able to support the objective to correct the failure; or ignore the problem. However this approach is not flexible enough since tactics constraint the ways a goal model can be modified. No conflict resolution mechanism is supported and the model does not provide ways to apply adaptations at runtime, by specifying the events and conditions that activate their execution.

Finally, a different point of view has been adopted by Morandini et al. [82], who refer to BDI agent models as reference architecture. The authors extend TROPOS to model the environment (the domain knowledge) and specify several types of goals and constraints between goals. The environment is represented in terms of UML classes, while the relation between goals is elicited through guard conditions between goal states and triggering transitions from a state to another. TROPOS has been extended with the possibility to model undesirable faulty states and recovery activities (e.g. activities to be performed, or new goals to add). Among recovery, the authors also propose a new relationship among goals called inhibition that works as follows: if goal A inhibits goal B, any time A has to be achieved, the achievement process of B has to be stopped until A is achieved.

3.4 Requirements for the definition of services

So far, the development of services has been mainly focused on their functionality and their integration with the existing software systems. This way the major design decisions are taken neglecting the main objectives the service must achieve. For this reason, Lau and Mylopoulos [83] adopt TROPOS model as a starting point for the design of web services. The authors associate each actor with a set of capabilities necessary to achieve its goals and to support the dependencies on other actors of

the model. Each capability must be also delegated to a specific agent in the system. Agents are modelled as web services, whose interface and fundamental data are described in a WSDL document, while agent capabilities are represented as WSDL operations. This approach is mainly focused on web services and does not offer the possibility to specify an interaction protocol (i.e., an order in which the service operations must be performed).

Other approaches [84,85] are more tailored to service compositions. For example, Lo and Yu [84] relate business models to "recurring" service-oriented solutions necessary for their realization. Business models refer to the actual design of business, such as the method of doing business, or a company business architecture. Business models are represented through i*, since they should express and deliver the vision and the objectives of the business, as well as model actor relationships and interactivity. Interactions among actors, devised in the business models, are realized concretely through business services that can adopt a recurring pattern. These patterns are collected in a catalogue to foster their reusability and can be implemented and executed via orchestration engines. Another important aspect of the design of processes is the definition of parameters for the selection of design alternatives depending on quality attributes or business goals. Unfortunately, current modelling languages, like BPMN, just provide workflow notations without the possibility to express a relationship between requirements and design alternatives. For this reason, Lapouchnian et al. [80] adopt a goal model to capture why a business process is needed and the many different ways how a goal can be attained. The goal model is augmented with control flow annotations to ease the generation of the workflow. These annotations can specify parallel or sequential goals, inclusive or exclusive OR-refinements, conditions, loops, event handlers or interrupts. They also model the input and output parameters of goals to determine resource requirements and the sequencing of goals. A specific alternative devised by an OR-refinement is selected depending on the preferences of the user or particular data/events. From the annotated goal model the authors generate the code of the corresponding BPEL process semi-automatically.

3.5 Variability in requirements for Product Lines

Within INDENICA it is important to understand how requirements for service platforms will vary. Thus, we discuss here how variation is considered on a requirements level. From a requirements perspective we can differentiate two main stages in product line engineering. These are the *scoping* step and the *full requirements* level description. The purpose of product line scoping is to identify the products and the amount of functionality that the product line shall cover. This boundary is then filled in terms of the full requirements description. Thus, effort in detailing requirements is not wasted on functionality that is not part of the final product line scope.

Product line scoping is akin to scoping in project management [86]. There, scoping has the focus on determining what should be part of the project and what should not be taken into account. In product line engineering we deal with a number of projects simultaneously, which leads to a split of the responsibilities of scoping [87]. Here, we will briefly discuss *product portfolio scoping* and *reuse infrastructure scoping*. Depending on the specific work, authors sometimes equate one or the other with scop-

ing in general (or address a third type: domain scoping). Surveys of scoping are provided in [87] and [88]. For example, when Clements [89] discusses the importance of scoping, he focuses only on the importance of determining the products that should be addressed as part of the product line. This view is strongly related to the issue of product management. In organizations that develop sets of products it must be determined which products should be developed. Along with this the set of supported requirements must be determined. This is discussed in [90] which discusses a comprehensive approach to product management. Scoping is also strongly influenced by economic considerations. This holds both for the product portfolio where the most beneficial products must be identified as well as on the reuse infrastructure level, where the most economic parts from a reuse perspective must be identified. Over time many different economic models for product lines have been developed [91]. A form of reference model has been proposed in [92]. This has also been extended into the SIMPLE-model [93].

Reuse infrastructure scoping on the other hand deals with the question: out of the functionality relevant to a product line as a whole, which parts should be developed in a reusable manner? Different approaches have been developed over time. The most comprehensive approach to date is still [94]. However, newer approaches exist. For example, [95] does also explicitly support the evolution of the scope.

A major point in the transition from the scoping stage to the full requirements stage is that while during scoping the product line (and its products) is in some form characterized through an enumeration of features at later stages explicit variability modelling is used. Thus, this transition also marks the transition between an extensional model and an intentional model. This also implies a significant generalization of the product line description.

While scoping is always done from the perspective of the product line as a whole, during full requirements engineering, it is more important to differentiate between domain requirements engineering and application requirements engineering [39]. While domain requirements engineering takes a perspective of the product line as a whole, application requirements engineering addresses the concerns of individual products or variants. As a consequence of this transition it is also important to observe how variability is integrated into product line requirements. In some approaches, in particular the early ones, there is no clear differentiation between the variability model and the domain model. Examples of this are FODA [25] and the HP approach [96]. A discussion of some early approaches to domain analysis is given in [97].

Meanwhile a different view has been established in product line engineering: variability modelling addresses the description of a model of how product configuration happens and is effective throughout the complete lifecycle. All the artefacts, including the requirements, can be configured on the basis of this model. This might be one of the reasons why on the one hand publications on domain analysis are becoming fewer and on the other hand requirements engineering publications sometimes cover explicit variability management.

An interesting special case where requirements cannot be clearly separated from variability is the issue of consistent description of requirements models for product

lines (i.e., consistency in all possible configurations). An example is [98]. This work focuses effectively on variation of requirements and discusses the use of general constraints (including numerical ones) for describing dependencies among them. The use of numerical constraints is, however, not new. Decision modelling approaches like [36] or work by Padmanabhan et al. [99] also used such an approach. Lauenroth et al. present an approach to determine whether requirements on the dynamic behaviour of systems remain consistent under variation [100].

Application requirements engineering takes the domain requirements (domain model) as input and aims to derive a specific characterization of requirements for an individual product or variant [39]. This activity poses significant challenges that require a clear differentiation of the various types of variability. A characterization of these types of variability is given in [101].

Ideally all requirements of the product can be described as a specific configuration of the domain requirements. However, while desirable, this will usually not be possible. Often product-specific requirements will be needed. A third category is introduced by Monzon [102]. In this approach, he proposes to differentiate between strong and weak derivation. Strong derivation corresponds to the configuration approach we outlined above, while weak derivation corresponds to a copy-and-adapt style. However, in general this approach (which is also called clone-and-own) is often avoided due to the maintenance problems it may incur.

Several authors also provide specific approaches that aim at application requirements engineering. Djebbi et al. [103] propose the RED-approach. This approach aims to support the decision which requirements from the domain model to use in a particular product. The matching of product requirements and domain requirements is explicitly supported. Adam et al. address a similar problem [104]. They focus on the question how to deal with requirements that fit the product line but have not been predicted in the domain requirements model [104]. Classen et al. propose a specific form of process support for application engineering [105]: feature configuration workflows.

As new products are developed and requirements change product line evolution occurs. A taxonomy of the different kind of requirements-driven evolution patterns is presented in [106].

3.6 Product Line scoping and RoI calculation

A major result in the domain of Product Line Engineering is the insight into the economical value of PLE and the calculation of the return on investment (RoI). A comprehensive derivation of the theoretical approach is given in [107]. The approach used in this work is called PuLSE-Eco and is the only approach that has been validated in industrial settings.

In [39] a practice-oriented guideline is given whereas [108] collects a summary of validations of the approach. The calculation of ROI for Product Line Engineering mainly results in a formula that contains the main influence factors and brings them into a structured relationship: The cost “C” for developing “n” versions of “X” product lines using reuse assets:

$$C_{org} + \sum_{i=1}^{n} C_{cab}(i) + X \times \sum_{i=1}^{n} (C_{reuse}(i) + C_{unique}(i))$$

- C_{org}: Cost for reorganization, process improvement, training, and similar topics
- C_{cab}: Cost for Core Asset Base development (incl. test)
- C_{reuse}: Cost for reusing core assets (e.g. adaptation, configuration)
- C_{unique}: Cost for product line specific development (incl. test)
- n: number of versions
- X: number of product lines using the same Core Asset Base

In [109] the authors add to the theoretical background a guideline for application of the approach. They suggest a Scoping Process that starts with a product line mapping in order to identify the planned products and their main features. Then they propose a domain potential assessment in order to verify that the organization is really ready for the transition to PLE. Finally a reuse infrastructure scoping is performed where the above-mentioned formulae can be adapted and applied for decision taking.

Based on experience in industrial environment at Siemens, Kreuter et al. [110] have enhanced the cost model by introducing several impact factors: Evolutionary versus revolutionary proceeding, small platform versus everything platform and proactive versus reactive platform.

A further summary of current practices and approaches in PLE scoping is given in [88]. They conduct a survey based on following factors: goal, method input, variability, method output, application domain, roles, effort, motivation/benefits, description level and maturity. In addition to these factors the “activity” is assessed which should indicate whether the method is probably used in practice. Applying these factors to a list of 16 published scoping methods their conclusions are:

- “Scoping has been established as an integral part of product line engineering”
- “Scoping approaches are distributed between different goals”
- “Some industrial applications exist”

But as a conclusion they also see that case studies and success stories are missing. A deeper survey with directly asking industry for input would be desirable, but industry is also reluctant, as product portfolio and product data are very sensitive and kept as company confidential information.

All mentioned approaches focus on the Product Line Engineering paradigm. In INDENICA the view changes to service platforms, which – from a cost model view – have similarities concerning re-use and domain specific assets. But as the flexibility of SOA is much broader more factors have to be taken into account. There is very little literature on these questions that show a promising approach.

4 Design

4.1 Methods and techniques for platform design

The engineering of virtual domain specific platforms requires a technical foundation that supports the construction of new functionality based on existing functional building blocks, also termed “modules”. Modularization has been an established method for system engineering for many years ranging from IT systems to embedded devices. The INDENICA platform needs to enable communication between these modules located in different layers of the Totally Integrated Automation (TIA) Pyramid shown in the figure below, ranging from the IT systems used in the Enterprise Resource Planning down to the controllers used in the Programmable Logic Controller (PLC) layer. This section examines the existing approaches for the design and runtime lifecycle phases, related projects and existing technologies at different layers of the TIA Pyramid.

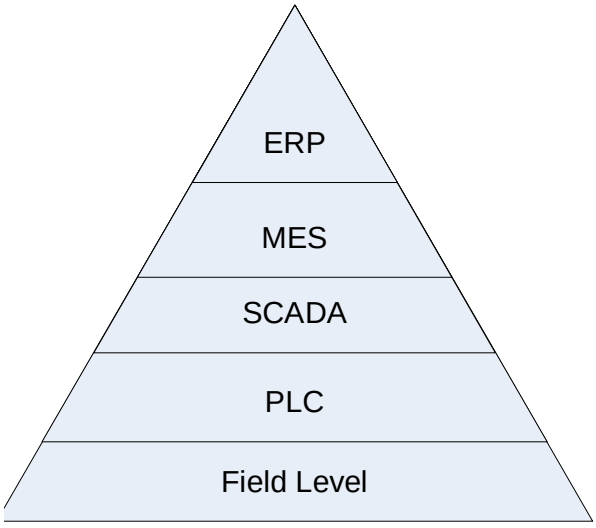

Figure 6: Totally Integrated Automation Pyramid

A usual design principle for platforms is to use a base technology for modularization and develop technical and / or domain services on top of these technologies. Different concepts and technologies like SOA, SCA, OSGi and Enterprise Service Bus (ESB) have been developed to support modularization and integration of functionality. Within each of the different layers the platform has to fulfil requirements that may vary: For example real-time responsiveness is usually not as critical in the ERP layer as it is in the PLC layer. The requirements to be met and the common functionality to be supported have to be elicited using the approaches described earlier in this document (see Chapter 3). Based on these requirements, a modularization technology that supports these requirements is chosen for the platform.

Aspects that usually need to be supported comprise governance of the service lifecycle, monitoring of service level agreements (SLA), composing new services based on existing ones, and enabling the communication between services as described by Josuttis [111] and Davis [112].

Regarding the design of technical and domain services, SOA and related modularization paradigms are state of the art in the upper layers (ERP and MES) of the automation pyramid. These paradigms imply that services are described using the Web Service Description Language (WSDL)[5]. Current researches in service platforms at this layer extend the concepts towards tradability and exchangeability of services. For this purpose the services are not only described using WSDL but additional non-functional aspects are described for example using the Unified Service Description Language[6] (USDL) developed to a large extend by the public funded project THESEUS TEXO[7]. One focus of this project has been to design a service engineering methodology called the "Integrated Service Engineering (ISE) Methodology" as described by Kett et al. [299] for creating business services based on the USDL and to host and trade them on a marketplace and therefore enabling the so-called service ecosystem [117].

To ease the deployment and trading of business services a platform was designed that provided common services: Amongst others a service catalogue, performance monitoring, service adaptation, billing and pricing, contracting, SLA management, access control and a generic hosting environment as shown in the following figure:

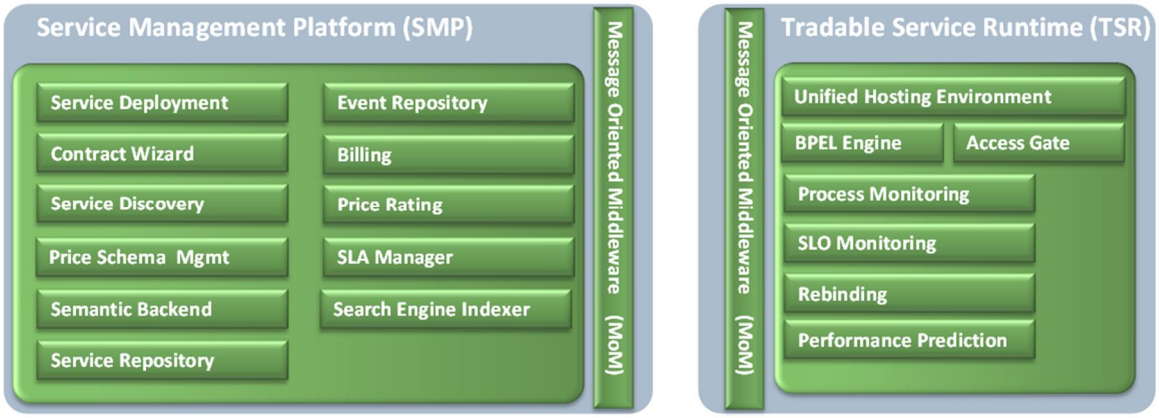

Figure 7: THESEUS TEXO Service Platform

The platform was logically divided between a Service Management Platform hosting common services related to management functionality and a Tradable Service Runtime that hosted all services necessary for the service execution. This design was due to the business model to be supported by the market place.

State of the Art for supporting variability in service ecosystems is the use of Business Process and Rule Engines, as shown in THESEUS TEXO e.g. for allowing different billing processes and flexible price schemata.

Usually, large enterprise applications are designed using a layered architecture framework, like the Zachman Enterprise Architecture Framework [301]. It allows a structured view and follows the idea of "separation of concerns": Perspectives of different stakeholders are supported as well as different abstractions like "data",

[5] http://www.w3.org/TR/wsdl20/

[6] http://www.w3.org/2005/Incubator/usdl/

[7] http://www.theseus-programm.de/en-us/theseus-application-scenarios/texo/default.aspx

"function", "people", "motivation", "time" and "network". To guide the development of services and the formal descriptions of their functional and non-functional features the Zachman Framework-based "Integrated Service Engineering (ISE) Framework" and the ISE Methodology described by Kett et. al in [299, 300] have been created that make use of model-driven software development (MDSD) technologies.

As outlined before the requirements to be supported by the platform differ with respect to the layer of the TIA Pyramid. Aspects like tradability of services are less relevant on device level (PLC). Here, for example real-time capabilities, reliability, availability and fast inter-service communication are of high importance. The previously introduced OSGi [5] framework (see chapter 2.2) additionally provides lifecycle management and dependency management but requires a Java Runtime Environment.

The following figure shows the container based on OSGi that has been developed for the OpenSOA communication platform of Siemens Enterprise Networks[8].

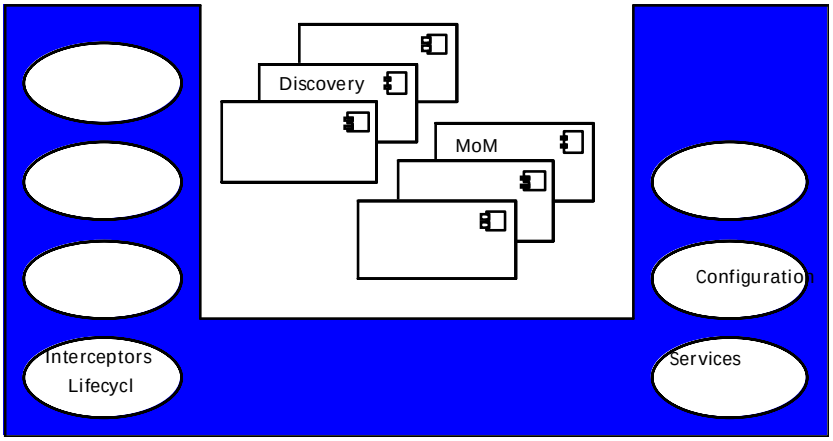

Figure 7: OSGi based OpenSOA container

It is evident that it has completely different platform services that provide common functionality to the services running on top of this container in comparison to the THESEUS TEXO platform.

A promising technology for integration of different platforms is the Service Component Architecture (SCA) that has been outlined before (see chapter 2.4). Implementations like Apache Tuscany[9] provide a wide range of bindings and therefore ease the development of service compositions based on services running in different platforms. Another implementation is the PocoCapsule[10] project develops an Inversion of Control (IoC) container for C/C++ platforms based on SCA.

The challenge of the INDENICA project is to identify the platform services at different layers of the TIA Pyramid, to provide formal descriptions for their configuration and

[8] http://www.siemens-enterprise.com/developerportal/Developer-Community/Resource%20Center/OpenScape-UC-App/OpenSOA%20SDK.aspx

[9] http://tuscany.apache.org/

[10] http://code.google.com/p/pococapsule/

usage and to enable compositions of services regardless of their logical location in the TIA Pyramid.

4.2 Decision models and patterns

Much work on better support for codifying the architectural knowledge (AK) has been done in the area of architectural decision modelling. Jansen and Bosch see software architecture as being composed of a set of design decisions [118]. They introduce a generic meta-model to capture decisions, including elements such as problems, solutions, and attributes of the AK. Another generic meta-model that is more detailed has been proposed by Zimmermann et al. [119, 120]. Tyree and Akermann proposed a highly detailed, generic template for architectural decision capturing [121]. A couple of other approaches are summarized in [122]. All these approaches share the problem of a significant extra effort necessary to record AK. INDENICA will address this problem by integrating AK recording with model-driven views, to easier enable a link between the models and decisions from which they originate.

Question, Options, and Criteria (QOC) diagrams [123] raise a design question, which points to the available solution options, and decision criteria are associated with the options. This way decisions can be modelled as such. Kruchten et al. extend this research by defining an ontology that describes the information needed for a decision, the types of decisions to be made, how decisions are being made, and their dependencies [124]. Falessi et al. present the Decision, Goal, and Alternatives framework to capture design decisions [125]. These approaches all make decision modelling more formal and precise; hence, we will integrate the experiences from these approaches in the design of INDENICA's view-based solutions. However, being more formal and precise than for example the decision templates by Tyree and Akermann [121] also means that these approaches require even more extra work to document the AK completely.

Recently, Kruchten et al. extended these ideas with the notion of an explicit decision view [126]. A decision view provides an addition and complement to more traditional sets of architectural views and viewpoints: it gives an explanatory perspective that illuminates the reasoning process itself and not solely its results. INDENICA follows this idea to combine the concepts of architectural views and AK to their mutual benefit. But INDENICA goes significantly beyond the approach by Kruchten et al. by defining not only the high-level views of the 4+1 view model [127], but also detailed technical views to allow for the model-driven generation of the software system.

INDENICA proposes to use architectural decisions to select patterns and hence ease the AK documentation by relying on pattern-based knowledge. Software patterns capture reusable design knowledge and expertise that provides proven solutions to recurring software design problems that arise in particular contexts and domains [128]. To systematically explain how to apply a number of patterns in combination, many pattern authors document patterns as part of pattern languages. A pattern language defines a collection of patterns in a domain and describes how these patterns can be combined [129].

Unfortunately, neither pattern languages nor architectural decision models solve all design and documentation problems for reusable design knowledge. For example, most practitioners only know a few patterns, such as the GoF design patterns [130], although the pattern community has documented patterns for many other domains. Hence, extensive upfront education is required to maximize the benefits that can result from using a number of patterns and/or pattern languages. In addition, decision modelling is most often done retrospectively. It is often seen as a "painful" additional responsibility without many gains [131, 121]. Different techniques, text templates, and tool support for decision modelling have been proposed, but failed to become broadly adopted in practice so far [120].

For these reasons, Zimmermann et al. [119, 120] propose reusable architectural decision models. In particular, Zimmermann et al. present a reusable decision model for recurring decisions in SOAs that is based on SOA patterns. A reusable architectural decision model is a decision model that is used to guide the architectural decision making activities [120]. Patterns and architectural decision models have many overlaps (for details see [131]). For instance, the approach of Zimmermann et al. uses decision models for pattern selection. The advantage of this approach is that a decision model which is based on patterns does not have to copy the pattern text and hence is easier to create than a self-contained decision model.

INDENICA will use these synergies of patterns and decision modelling to ease the decision modelling process for virtual service platforms. INDENICA will link the decision models to view-based models in order to link decisions and the models that created following the decisions.

4.3 Modelling variability in architecture

In this section, we first explain general aspects how variability can be supported by an appropriate architecture. This is done on a conceptual level. Afterwards, we discuss approaches for identifying the right architecture in the context of product lines.

On an abstract level, there are three basic ways to realize variation in an architecture [20] (Figure 8): adaptation, replacement and extension.

- In the *adaptation technique*, there is only one implementation available for a certain component, but it offers interfaces to adjust its behaviour in an object-oriented manner.
- In the *replacement technique*, several implementations of a component are available. Each implementation supports a different product variation.
- In the *extension technique* only an interface is given. In order to implement variability, new components must be developed and bound to the interface.

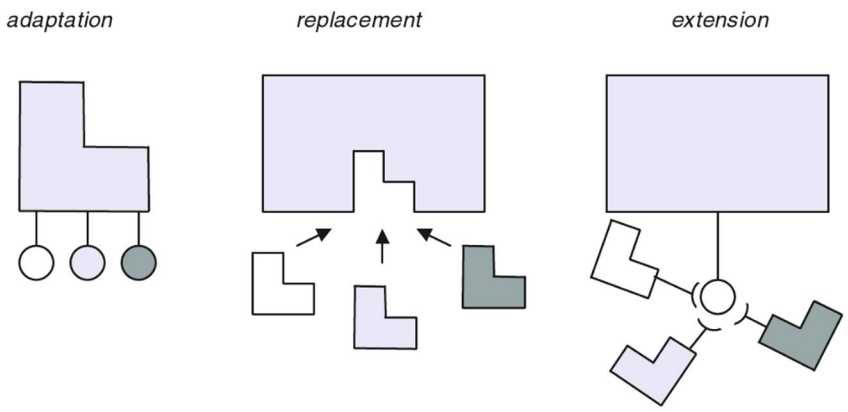

Figure 8: Three basic techniques for realizing variability in architecture [20].

For facilitating variability at implementation level, an appropriate architecture is needed. There exist various techniques for modelling architectures with variability support. These methods can be classified into two different categories: architecting methods that aim specifically at product lines and single-system methods that could successfully be used to model software product lines.

Single-system methods that have been successfully applied in the context of software product lines are for instance the Attribute Driven Design (ADD) method [132] and the work of Jan Bosch [133].

Matinlassi [134] compared five different techniques for modelling software product line architectures: COPA [135], FAST [136], FORM [31], KobrA [137] and QADA [313]. The result was that the compared concepts are not competing with each other. Instead, every technique has different focus as follows [134]:

- COPA: Covering all aspects of product line engineering i.e. architecture, process, business and organization, whereas this method is concentrated on balancing between top-down and bottom-up techniques.
- FAST: Family oriented process description with activities, artefacts and roles. Therefore it is very interesting but not applicable as it is.
- FORM: Feature-based approach for capturing commonalities and variabilities inside a domain. Covers also architecture design and development of code assets.
- KobrA: Simple method for traditional component-based software engineering with UML.
- QADA: Focus on architectural design according to quality requirements.

Other important techniques for modelling product line architectures that were not included in [134] are PuLSE-DSSA [139, 140] and Koala [308, 309]. Modern refinements of Koala include MontiArc^{HV} [308] and Δ-MontiArc [311] which also supports hierarchical variability during the architectural design.

The PuLSE-DSSA (Product Line Software Engineering – Domain Specific Software Architecture) methodology developed by the Fraunhofer Institute for Experimental Software Engineering (IESE) [139, 140]. PuLSE provides a complete framework that covers the whole software product line development life cycle. PuLSE-DSSA uses the

input given from scoping and domain analysis to model a reference architecture for the software product line with means of object-oriented frameworks [141].

An often referenced example is the Koala component model [308]. This component model supports the integrated specification of variability with the architecture view. Product Populations [309] extend the Koala component model and facilitate the use of variable components across several product lines. Furthermore, the Koala component model has been extended to support also hierarchical variability in [322] and more recently in MontiArc^{HV} [308]. The later, however, also emphasises the use during design time. MontiArc^{HV} [308], Δ-MontiArc [311], and Plastic Partial Components [312] extend this approach to provide an architecture discribution language (ADL), which can be used during architectural design.

4.4 Architectural views

The concept of architectural views is not new. However, to the best of our knowledge, there are only few approaches that exploit the notion of views to support business process modelling and development. [142] presents a view integration approach inspired by the idea of schema integration in database design. Process models based on Event-Driven Process Chains (EPCs) [143, 144] are investigated, and the predefined semantic relationships between model elements such as equivalent, sequence, and merge operations are performed to integrate two distinct views. Semantics-based merging is a promising approach to model integration, but it is hard to apply it in order to integrate two different types of models, for instance, to merge a control flow model with a data model. Thus, the authors mainly focus on the merging of process behaviours (i.e. the control flow).

The AMFIBIA approach [145,146] presents a meta-model for formalizing different aspects of business processes and provides an open framework to integrate various formalisms through a central notion of interface. The major contribution in AMFIBIA is to exploit dynamic interactions of those aspects. AMFIBIA's framework has a core model with a small number of important elements, which are referred to, or refined in other models. The distinct point to our framework is that in AMFIBIA the interaction of different 'aspects' is only performed by event synchronization at run-time. Using view extension and integration mechanisms in our framework, the integrity and consistency between models can be verified earlier at the model level. Nonetheless, AMFIBIA offers no support for the separation of abstraction levels and adaptation to stakeholders' interests.

The ISO Reference Model for Open Distributed Processing (RM-ODP) [147] is a standard reference model that defines a set of different viewpoints such as enterprise, information, computational, engineering, and technology. Each viewpoint has its own language and clear semantics. These concepts, similar to those in AMFIBIA, are defined based on the principle of separation of concerns to help stakeholders thinking from different perspectives in order to manage complexity of distributed applications. The advantage of our approach compared to these approaches is that our view-based model-driven framework does not only support the separation of process concerns but also the separation of process models into different levels of abstraction, for instance, abstract and technology-specific layers.

IDEF3 [148] is a scenario-based framework proposed for modelling business processes. IDEF3 provides two fundamental views: the *process-centered* and the *object-centered* view. The process-centered view provides a graphical representation that supports domain experts and analysts in describing business processes with respect to events, activities, and their relationships. The object-centered view is a mean for capturing information about objects of various kinds and their transformations during the course of a particular process. The other process concerns such as service and process interactions, transactions, event handling, etc., have not been considered in IDEF3.

In the following, we briefly introduce the View-based Modeling Framework (VbMF) [149,150] that is the conceptual foundation for our work in WP3. However, as VbMF focuses on business processes, a novel approach will have to be implemented in INDENICA, which follows the general approach and significantly extends the one used in VbMF. VbMF exploits the notion of views to separate the various process concerns of a business process in order to reduce the complexity of process-driven SOA development and enhance the flexibility and extensibility of the framework. VbMF offers a number of modelling artefacts, such as view models and view instances (or views for short) organized in two levels of abstraction. Each view embodies a number of view elements and their relationships that represent a business process from a particular perspective. View elements and their relationships are precisely specified by a view model. In other words, a view model is a (semi-)formalization of a particular process concern and the views conforming to that view model are concrete instances of the process concern.

VbMF initially provides three foundational (semi-)formalizations for representing a business process which are the *FlowView*, *CollaborationView* and *InformationView* models. The *FlowView* model describes the orchestration of process activities, the *CollaborationView* model specifies the interactions with other processes or services, and the *InformationView* model elicits data representations and processing within processes as well as messages exchanges.

However, VbMF is not merely bound to these view models but can be extended to capture other concerns, for instance, human interaction, data access and integration, transactions, and fault and event handling, have been realized in the past. VbMF view models are derived from fundamental concepts and elements of the Core model. Therefore, these concepts of the Core model are the extension points of the view-based modelling framework [149,150]. In INDENICA we will extend this basic idea with novel, more flexible view concepts and views that are tailored to the concerns in INDENICA.

4.5 Model-driven service composition

Previous research in the area of Web service composition models has involved the adaption of traditional formalisms such as process algebras, temporal logic, Petri Nets, etc., for Web service composition. For instance, Hamadi and Benatallah [151] used Petri Nets to construct Web service compositions with specific, formally verifiable properties. Margaria et al. propose a novel technique that harmonizes a variant of linear temporal logic (LTL) for formalising service compositions and situation cal-

culus for planning the composite services [323]. The approach presented by Naujokat et al. in [324] aims at supporting the automatic synthetizing of business processes given a set of descriptions of services being used in the processes. Recently, a considerable amount of work on semantic-based approaches for service compositions and business processes have been reported by Petrie et al. in [325].

Languages and models on higher levels of abstraction (such as UML, or BPMN) were analyzed by other researchers who found that these models, in addition to being easier to write and interpret, can also be used to express the most important flow patterns [152]. These languages are therefore a natural choice for researchers aiming at bridging the gap between business workflows and models on the one hand, and technical service compositions on the other. The approach of specifying compositions on business model level is called Business Driven Development (BDD) [153], following the MDD naming.

UML has been used as a language to model Web service compositions by Skogan et al. [154]. They propose a method that utilizes UML Activity models to define compositions, and an MDA-based approach to generate executable code in varying composition languages. They have produced an initial prototype that supports the generation of both WS-BPEL and WorkSCo executable code. Their method endorses a complete round-trip model: existing WSDL specifications are refactored into UML models, which in turn are arranged to create new service compositions; these compositions can then be transformed into executable code (either WS-BPEL or WorkSCo), and deployed on a workflow engine. They use UML as an integration platform, which is independent of the actual execution environment. The actual transformation of high-level UML models to workflow code can be handled using XSLT transformations; this is possible since both WS-BPEL and WorkSCo are represented using XML.

The approach of Skogan et al. has later been extended to also consider semantic Web service composition and QoS attributes [155]. The main advantage of this extension is the increased support for run-time discovery of services and run-time service binding: in [154] the service bindings (the partner links in WS-BPEL notion) are defined at design-time, while [155] considers abstract bindings to specific semantically defined functionality; concrete services implementing this abstract functionality are discovered at run-time using semantic discovery and matchmaking technologies. Additionally, services are ranked based on QoS properties, i.e., if more than one service instance is discovered during run-time, the one exhibiting the best QoS properties is selected. Unfortunately, the work is based on a set of assumptions (e.g., the existence of a semantic matchmaking and service discovery entity) that are not fulfilled today.

Koehler et al. have presented an approach that is more clearly aligned to the OMG MDA [156] and BDD as early as 2003. Their work covers the transformation of business models (represented in ADF and UML) to technology-level WS-BPEL code. In contrast to other approaches, their work considers not only top-down transformations from high-level models to code, but also considers the bottom-up reengineering of existing WS-BPEL compositions into business processes. Business models are first transformed to process graphs, which are in turn refined to flow graphs; these subflows can then be compiled to solution components using plat-

form-specific transformations (synthesis). In the reverse direction (bottom-up), solution components are combined to create flow graphs, which are again merged to process graphs, and finally the business model can be restored. Later, the same authors have refined their ideas in [157], detailing how graph-based process models can be transformed into executable code using graph transformation and compiler theory.

Ouyang et al. use a different language, OMGs BPMN, to represent business models [158]. They argue that BPMN is (even though the language is relatively young) more common in the business world than the more software-centric UML. Additionally, BPMN is well supported by business analyst tools. However, the authors state that the transformation from BPMN to BPEL is complex since BPMN is (as a graph-oriented language) fundamentally different than the block-oriented WS-BPEL. However, they argue that their approach still is complete, fully automated and produces WS-BPEL code that is comprehensible for humans.

A model-driven service composition approach based on composition rules is introduced in Orriens et al. [159]. The composition rules are expressed in OCL (the Object Constraint Language), and are used to model constraints on possible compositions. Examples of such rules are `activity.function = "FlightTicketBooking"` (i.e., the function of the activity always has to be "FlightTicketBooking") or `activity.input -> notEmpty()` (i.e., there always has to be an input message for this activity). Generally, they distinguish between structural rules (structuring activities with the composition), data rules (rules that relate messages to each other), behavioural rules (rules that guard the integrity of the composition, and preconditions, postconditions and invariants of activities), resource rules (rules that guide the usage of resources) and exception rules (rules that specify the behaviour of the composition in exceptional cases). When all necessary rules that apply for a composition are specified, an automated dynamic service composition engine is used to create a composition that satisfies the rules specified by the model. However, this approach naturally implies that the composition has to be fully specified using business rules.

5 Adaptation and governance

5.1 Variability at implementation level

The variability that is modelled must also be implemented. Therefore variability implementation has received significant attention over the years [160, 161]. We will first introduce general variability implementation techniques and the corresponding concept of binding times; then we will concentrate on runtime variation and Dynamic Software Product Lines (DSPL), which we expect to play a crucial role in the context of the INDENICA project as they also address variation for service platforms.

For the realization of variability, there exist several techniques. These techniques can be categorized according to the realization of variation before, during, or after compilation, so called binding time. The choice of binding time is independent of variability modelling [39].

Most work focused on development-time system composition (for example, there are some professional tools [314, 315]). Aspect-oriented programming and model-based development have also been used as a basis for variability implementation [162]. Few approaches, however, have addressed run-time composition (e.g., web-service technology), and even less work has addressed the problem of transparently moving the resolution of variability between development time and runtime. Notable exceptions include the works by van der Hoek or Schmid [163, 164]. These approaches can also be used to exchange different variability implementation technologies.

Today, a product in a product line must adapt to its environment, cope with unforeseeable resource constraints and interact with other systems for which its designer could not have adjusted it at development time. It is impossible to foresee all functionality or variability a software product line might require at runtime.

Designing for runtime variation gets especially important when independent software systems or components are newly coupled at runtime, as it frequently occurs with services in service-oriented computing. Service platforms like INDENICA provide flexibility at runtime as they support applications in dynamically exchanging service implementations. Thus, product line based approaches to runtime variation like Dynamic Service Product Lines [165, 166] are of particular interest here. They give more room to monitoring the current situation and planning appropriate adaption than in the traditional SPL approach. Also some other work on adaptive systems explicitly relies on product-line engineering, such as [167] and [168].

5.2 Adaptation frameworks

Adaptation of service-based systems has been a more than active research topic for the last years. Many groups and individuals have proposed different approaches towards adaptation of service-based systems, which are considering adaptation on a variety of levels.

5.2.1 Types of adaptation

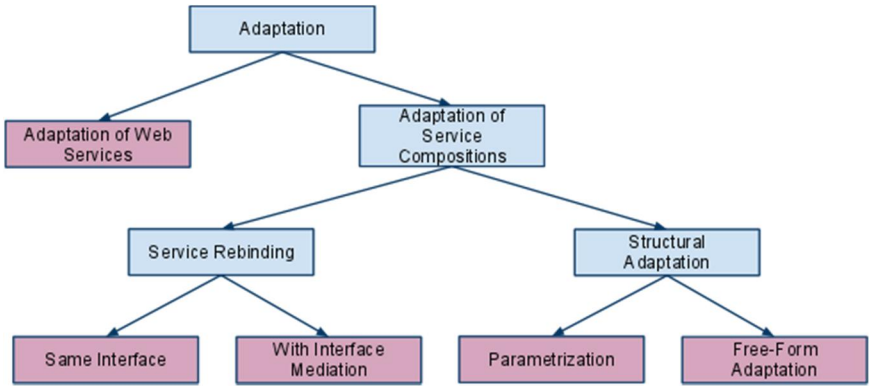

Figure 9: Taxonomy of Adaptation

Figure 9 summarizes different types of adaptation in service-based systems. Primarily, one can distinguish between adaptation of (atomic) Web services and service compositions. The former is in fact not very different to adaptation of any other software artefact, hence little research specific to this task has been carried out. One of the few examples is [169], which proposes an aspect-based approach to adapting service interfaces. The latter, adaptation of Composite Services, can again be divided into two separate classes of adaptation: service rebinding and structural adaptation.

Historically, service rebinding is the more traditional (one can also say 'older') type of adaptation. Specifically, service rebinding without interface mediation (that is, rebinding of services with identical WSDL interfaces) can be seen as state-of-the-art by now. Research approaches that consider rebinding-style adaptations include PAWS [170], WS-Binder [171], MASC [172], VieDAME [173] and VRESCo [174]. These approaches, as well as many others not explicitly named here, share the common characteristic that they add something on top of just exchanging one service endpoint for another (which is in fact already supported by the WS-BPEL specification itself through the dynamic partner link facilities). For instance, VRESCo and VieDAME have a strong focus on selecting the 'best' service for rebinding based on known Quality of Service information. Furthermore, VRESCo can mediate between services that provide the same functionality using different technical interfaces. Similar facilities are also provided by WS-Binder. In MASC, the adaptation process is guided by a powerful policy-based decision making component. PAWS goes one step further, and utilizes rebinding for self-healing and automated failure recovery (but does not cover mediation of different service interfaces). [175] discusses the problems arising from the requirement of runtime adaptation and present the solution of a seamless replacement of the service implementation at execution time in a service-oriented component model. This approach was based on CoRBA (Component Based Runtime Adaptable) and used dynamic Aspect-oriented Programming (d-AOP) to provide adaptation without interference of the application execution and the service availability.

In recent times, more research interest has been directed towards structural adaptation of service compositions. Structural adaptation is different from rebinding in that not only the base services of the composition are changed, but the structure of the composition itself is adapted. This can include, but is not limited to, adding activities such as service invocations, removing activities, changing branching or looping condi-

tions, or refactoring parts of the composition into a new sub-composition. A more comprehensive view of possible patterns of structural adaptation has been presented in [176]. One relatively simple approach to structural adaptation is parameterization [177]. More complex structural adaptations, specifically adaptations that have not been predefined at composition design time, have proven to be quite hard to model and implement. One approach that has been adopted by various different groups is the notion of adaptation based on the concept of aspect orientation. Probably the first prototype going in this direction was AO4BPEL [178]. However, AO4PBEL is not strongly focused on the runtime adaptation of compositions, but more on modelling compositions in an aspect-oriented way. [172] proposes a novel service composition Framework called AdaptiveBPEL. The framework enabled service composition for dynamic change in order to provide a greater degree of configurability and dynamic adaptation of Web Services. In this approach authors presented adaptation process that was policy-driven in order to provide dynamic and client-specific customization of Web Services. AdaptiveBPEL is closely related to the MASC system presented above. Another example of Aspect-Oriented approach was presented in [179] which focuses on adaptation of composite Web Services in response to changes in non-functional requirements. A later approach dubbed BPEL'n'Aspects [180] was more geared towards runtime adaptation (and also improved on some other limitations of AO4BPEL). Finally, the PREvent framework (initially presented in [181]) also includes means for structural adaptation based on the aspect-oriented paradigm [182]. Unlike the approaches mentioned before this work is strongly geared towards adaptation for preventing violations of service level agreements.

5.2.2 Further challenges in adaptation frameworks

On a more strategical level, [183] propose and evaluated an application-transparent adaptation strategy for SOA-based systems. The formal adaptation model derived from a suite of metrics enabled focusing on efficiency in terms of performance, memory, network, and processor utilization by adapting to the changes in the environment. [184] surveys representative software adaptation methods and proposed four types of service variability: workflow, composition, interface and logic. In the same paper authors proposed a framework for Service adaptation based on adaptation patterns. As shown in the paper, the framework is capable of greatly increasing of usability and applicability of the adapted services. Adaptation of Web Service Composition using Expiration Times was presented in [185]. The measurements of the impact of expected QoS changes in the Service parameters were used to make an optimal adaptation decision. This approach focused on the efficiency of the adaptation that is implied when using dynamic process models that was incurring the least cost possible. Another approach that leveraged Aspect-oriented programming was introduced in [186] where the main focus was on the scope in which Web service adaptation is required. Authors proposed to discuss adaptation in the scope of context, device and customer (service customization) with a single methodology – WS-Adaptation. In this approach authors facilitated model-driven approach and showed the need to provide server-side adaptation in opposite to client-side presented in the most research papers.

Very valuable work in the area of service monitoring and adaptation has been done in the EU FP7-funded project ENTHRONE. [187] describes the part of this project that was mainly focused on providing end-to-end Quality of Service (QoS) guarantees in multi-domain environments. The project presented a service monitoring system that provided monitoring information to service providers. The aims were to provide quantified QoS-based services, to enable dynamic adaptation, and to network operators for making provisioning decisions and allowing dynamic resource allocation for optimizing the usage of network resources.

In the context of adaptation, one important aspect is mediation among incompatible services. In [188] authors propose an adaptation machine that sits between pairs of services and manipulates the exchanged messages according to a repository of mapping rules. The paper formulates an operational semantics for the adaptation machine, including algorithms to compute rule firing sequences and criteria for detecting deadlocks and information loss. The adaptation machine acts as an adapter between incompatible services: it selects and chains mapping rules to resolve mismatches as they arise. The same problem was addressed in [189] where authors present a service behavioural adaptation approach that is based on dependency graph. The problem of behavioural adaptation is divided into three sub-problems: service description (as a foundation of adaptation), service mismatch definition and adaptor construction process. The approach assumes detection of all possible mismatches and generation of different adaptors correspondingly. [190] presents an approach that leverages Enterprise Service Bus (ESB) mediation features to provide dynamic adaptation capabilities within the service infrastructure layer. The two key motivations for such an approach were the ability to respond to dynamic business requirements of service provider and the need to fulfil QoS requirements specified in Service Level Agreements (SLA). In order to realize self-adaptive behaviour authors proposed that system is equipped with control loops that collect relevant information, from the system and its environment, and act accordingly to pre-defined adaptation strategies. The work described in this paper was a part of the S-CUBE network – funded by EU FP7. [191] is another paper from the adaptation area that was funded by S-CUBE project. Authors propose an adaptation strategy framework to support the design of Service Based Applications (SBAs) that targets the adaptation requirements raised by context changes. Context in the adaptation activities is described in a model.

Applying adaptation strategies that are usually applied as a set of patterns and policies also needs to take care of dynamism of configuration as a whole. The paper [192] describes the concept of software adaptation patterns that are described in terms of a three-layer architecture for self-management. A software adaptation pattern defines how a set of components that make up an architecture pattern dynamically cooperate to change the software configuration to a new configuration. In this approach, adaptation connectors are introduced to encapsulate adaptation state machine models so that the adaptation patterns can be more reusable. The adaptation patterns presented in this paper are part of the Self-Architecting Software Systems (SASSY) model-driven framework for runtime self-architecting and rearchitecting of distributed service-oriented software systems.

5.3 Service platform governance

Before concentrating on the governance of service platforms, we can start with a general definition. Wiktionary[11] says that governance is:

- The process, or the power, of governing
- The specific system by which a political system is ruled
- The group of people who make up an administrative body
- The state of being governed
- Accountability for consistent, cohesive policies, processes and decision rights

ITIL[12] gives a more specific definition with IT relevance: "Ensuring that Policies and Strategy are actually implemented, and that required processes are correctly followed. Governance includes defining Roles and responsibilities, measuring and reporting, and taking actions to resolve any issues identified."

[193] adds the aspect of decision making and the community aspect: "Governance is the process of making correct and appropriate decisions on behalf of the stakeholders of those decisions or choices [...] Governance is ensuring that behaviour conforms to norms, expectations and guidelines set forth by the community or elected leadership of a community."

5.3.1 Corporate governance

Corporate governance[13] is the set of processes, customs, policies, laws, and institutions affecting the way a corporation (or company) is directed, administered or controlled. Corporate governance also includes the relationships among the many stakeholders involved and the goals for which the corporation is governed. The principal stakeholders are the shareholders, the board of directors, employees, customers, creditors, suppliers, and the community at large.

In [193] the definition of Corporate Governance is directly and only related to processes: Corporate Governance is the process of ensuring the best interests of a company's or organization's stakeholders are met through all corporate decisions, from strategy through execution; engagement of critical stakeholders in key decisions of an organization.

5.3.2 IT governance

IT Governance[14] is a subset discipline of Corporate Governance focused on information technology (IT) systems and their performance and risk management. The rising interest in IT governance is partly due to compliance initiatives, for instance Sarbanes-Oxley in the USA and Basel II in Europe, as well as the acknowledgement that IT projects can easily get out of control and profoundly affect the performance of an organization.

[11] http://en.wiktionary.org/wiki/governance

[12] http://www.itil.org/en/glossar/glossarkomplett.php?filter=G

[13] http://en.wikipedia.org/wiki/Corporate_governance

[14] http://en.wikipedia.org/wiki/IT_governance

For IT Governance reference models and standards are in place or being developed, namely ITIL, COBIT and ISO20000:

ITIL (Information Technology Infrastructure Library) describes a systematic, professional procedure for the management of IT services. The library emphatically puts the emphasis on the importance of meeting the corporate requirements from the commercial aspect. IT service management under ITIL is geared purely towards customer benefit and efficiency. Achieving the business objectives whilst simultaneously meeting internal and external requirements is fundamental to ensuring a company's medium and long-term success.

The COBIT framework is aimed primarily at compliance and security and, as such, ensures the IT governance for the operation of the IT services. This best practice framework supports the control of all IT-processes and is primarily geared towards auditing and ensuring compliance. The synergy between the two approaches (ITIL and COBIT) lies in the fact that the more formal control objectives of COBIT are being aligned with the ITIL framework which is orientated towards suitability and flexibility and these must be fulfilled in a way that can be defined.

ISO 20000: The two frameworks (ITIL and COBIT) will continue to develop and increasingly converge, with the bridge for this being created by the international ISO 20000 standard. Based on ITIL the two organizations itSMF and BSI (British Standard Institute) have developed this clearly measurable standard and therefore created the opportunity for certification of the conformity, effectiveness and efficiency of the individual IT service management control objectives.

5.3.3 SOA governance

Wikipedia[15] gives a vey generic definition: "Service-Oriented Architecture (SOA) governance is a concept used for activities related to exercising control over services in an SOA. SOA governance can be seen as a subset of IT governance which itself is a subset of Corporate governance. [...] SOA requires a number of IT support processes as well as organizational processes that will also involve the business leaders. [...] based on standards and includes policies, contracts and service level agreements."

[193] is more focusing on the stakeholders and their involvement: "SOA governance is the ability to ensure that all of the independent (SOA) efforts (whether in the design, development, deployment, or operations of a service) come together to meet enterprise requirements. SOA governance is the process of ensuring all business and IT stakeholders' interests are served by the planning, funding, and execution of an enterprise SOA initiative".

And [193] again highlights the importance of decisions: "SOA governance is the definition, implementation and ongoing execution of an SOA stakeholder decision model and accountability framework that ensures an organization is pursuing an appropriate SOA strategy aligned with business goals, and is executing that strategy in accordance with guidelines and constraints defined by a body of SOA principles and policies."

[15] http://en.wikipedia.org/wiki/SOA_Governance

[194] goes one step further and makes a distinction between design-time and runtime governance: "SOA governance provides a set of policies, rules, and enforcement mechanisms for developing, using, and evolving service-oriented systems, and for analysis of the business value of those systems. SOA governance includes policies, procedures, roles, and responsibilities for design-time and runtime governance.

- Design-time governance includes elements such as rules for strategic identification, development and deployment of services; reuse; and legacy system migration. It also enforces consistency in use of standards, SOA infrastructure, and processes.
- Runtime governance develops and enforces rules to ensure that services are executed only in ways that are legal and that important runtime data is logged. From a life-cycle point of view, design-time governance applies to early activities such as planning, architecture, design, and deployment. Runtime governance applies to deployment and management of service-oriented systems."

The runtime governance definition directly leads to "compliance" which will be discussed below.

5.3.4 SOA governance reference models

In parallel to the development of SOA and the pervasion of SOA in enterprises, SOA Governance has developed. It is frequently seen as a sub-discipline of IT Governance (see definitions). A scientific approach to SOA Governance came up only recently.

Some contributors try to set up a SOA Governance model based on existing standards: For instance [195] take a holistic approach to cover all aspects of governance based on the OASIS SOA Reference model [196].

[197] tries to create a reference model based on COBIT. In fact they also rely on ITIL when it comes to defining the structure of their model. While referring to COBIT their focus is very much oriented towards evaluation and improvement processes. [198] also sketches a governance model, but take a dedicated regard to web service technologies and their implications in the context.

SOA.com[16] is a commercial tool provider, but their definition of integrated SOA governance also gives a well-defined structure similar to a reference model. It "… ensures the applicability, integrity and usability of a wide range of assets through all their lifecycle stages from asset identification through deprecation. The full lifecycle is split into planning governance, lifecycle governance, and operational governance, with a crosscutting policy governance theme.

- Planning Governance includes identification, analysis and modeling of candidate services, policies, profiles, processes and information. An effective planning governance tool manages an organization's SOA portfolio: It enables examining existing and planned applications and determines which capabilities should be ex-

[16] http://www.soa.com/solutions/integrated_soa_governance/

posed as services and where applications would benefit from consuming shared services.

- Development Governance marshals an asset through the typical design through deprecation phases of its lifecycle. It typically includes a workflow mechanism to approve migration, policy compliance validation, and a clear separation (logically, physically, or both) between lifecycle stages. Development Governance is the realm that most registry vendors have moved towards.
- Operational Governance controls the runtime aspects of SOA. It typically includes service monitoring, security and management with a runtime policy system. Most Web Services Management vendors now position themselves as providing operation governance solutions.
- Policy Governance defines and manages policies, associates them with various assets, and validates and reports on policy compliance. "

The most comprehensive description of a SOA Governance Reference Model is given in [193]. This book describes in detail all the implications for strategy (Figure 10) building down to enabling technologies and tools. It also has a strong organizational focus and is designed to help industrial and public companies to transform its classical IT to a SOA-based IT in a structured way.

For runtime governance monitoring it defines requirements and implementation guidelines, but it does not select any technical solution components nor does it define an architecture.

Figure 10: SOA Governance Reference Model by Marks

Common to all above-mentioned approaches is the fact that they do not go into details when it comes to the question of monitoring service performance and service level agreements. [198] proposes a SOA based governance model to handle non-functional requirements. They define SOA governance objectives and then derive requirements for a SOA governance model that again consists of an operational model, an object model and a management model.

5.3.5 SOA governance frameworks

The more we come down from Reference Models to Frameworks the fuzzier become the terms. On one side the initiative of The Open Group to standardize terms, structure and procedures in SOA Governance is called framework [199].

Niemann et al. [200] summarize the current research on SOA Governance (Figure 11) and suggest a generic structure for a SOA Governance Framework. They give an overview on existing frameworks – from literature and from industrial product offerings. It shows that none of the approaches cover all aspects of SOA Governance.

In their further work they structure the elements of a SOA Governance Model: 1) the SOA goals, 2) SOA as enterprise architecture, and 3) the control cycle. What they do not yet take into account is the SOA Governance Framework developed by The Open Group [199]. This takes a very ambitious and industry-driven approach and is heading for standardizing a model. It is published as a draft specification but as of today it is not a standard yet. This is a shared work of a vendor-neutral and technology-neutral consortium and is currently a draft for a standard made available for comment. Even though it might change during the review process it is worth having a closer look here.

Legend: ● — proposed and integrated ○ — partially integrated	Impact on organization	SOA Maturity Model	New roles & accountabilities	Best Practices	Metrics model	Impact on people's behaviour	SOA lifecycle	SOA roadmap	Policy catalog	Service lifecycle	Governance processes	Policy enforcement mechanisms
Brauer and Kline (2005)	-	○	-	-	-	-	-	●	○	●	-	-
Bieberstein et al. (2005, 2006)	●	-	●	●	-	●	-	●	○	-	●	-
WebMethods (2006)	○	-	-	-	-	-	●	-	●	●	-	●
Software AG (2005)	●	●	●	●	-	-	-	●	●	●	○	-
BEA Systems, Inc. (2006)	-	-	-	-	-	-	○	-	●	●	-	-
SAP AG (2004, 2006, 2007)	●	-	○	-	○	-	-	-	●	●	○	●
Afshar (2007), Oracle	●	●	●	●	-	●	-	-	●	●	-	-
IBM (2006)	●	-	○	●	○	○	●	-	○	●	-	-
Marks and Bell (2006)	●	-	●	●	●	●	●	○	●	○	●	●
Schelp and Stutz (2007)	●	-	●	-	-	-	-	-	-	-	○	-
Allen (2008)	●	●	●	○	-	○	-	-	●	●	●	-

Figure 11: Overview on SOA Governance Frameworks, [200]

This framework consists of a SOA Governance Reference Model (SGRM) and a SOA Governance Vitality Method (SGVM). The SGRM defines a comprehensive structure containing guiding principles (policies), governing processes, governed processes, artefacts, roles and responsibilities and technology. The vitality method is a guide-

line for tailoring the model and for introducing SOA governance in an organization. It sticks to the well-known improvement cycle Plan – Define – Implement – Monitor.

5.3.6 Governance compliance and runtime monitoring

All above listed Governance Models and Frameworks listed above refer to compliance as a strong need of business and organizations. In the SOA Governance Framework by The Open Group compliance is seen as one of the three SOA governing processes [199]. But it does not go beyond the suggestion to insert checkpoints in the defined SOA processes.

The FP7 funded project COMPAS did a deep diving here and defined a compliance-driven architecture for services. An introduction is given in [201]. Their definition of compliance governance is "Compliance governance refers to the overall management approach for controlling the state of compliance in the entire organization and, in general, consists of: (1) selecting the sources to be compliant with and designing corresponding compliance requirements; (2) (re-)designing business processes compliant with the selected requirements; (3) monitoring compliance of processes during their execution; (4) informing interested parties (managers, auditors) on the current state of compliance; (5) taking specific actions or changing the processes in cases of (predicted or happened) non-compliance".

The COMPAS compliance-driven governance architecture describes the components, events and messages that allow monitoring the business process execution at runtime, tracing compliance violations for later root cause analysis, collecting relevant data for reasoning and displaying the compliance status and KPIs on the compliance governance dashboard.

What is left open for further research is the definition of Key Compliance Indicators, the mechanisms for reacting on non-compliance and the roles involved in decision making. In the scope of INDENICA this becomes even more complex due to the virtualization of platforms and varying compliance requirements in the different technical platform domains.

5.3.7 SOA governance tool suites

In addition to the theoretical frameworks numerous IT infrastructure and tool providers collect their SOA related products and services and call this offer also “SOA Governance Framework”. The aim of this section is to give a short overview on tools and frameworks that support SOA governance adoption within enterprises. The overview is not a comprehensive market study but rather a listing of main suppliers and their main features. Evaluation of capabilities and suitability for virtualized service platforms will be subject to further work. A good overview is given in [194].

Soa.com's Integrated SOA Governance[17] promotes the core SOA governance best practices of:

[17] http://www.soa.com/solutions/integrated_soa_governance/

- Governance Automation - lifecycle management workflow to implement building permit process, integrated provisioning and lifecycle management, and inter-departmental contract management and negotiation
- Uniform Policy Management - uniform lifecycle and policy governance across existing platform investments
- Meta-data Federation - seamless, heterogeneous SOA Governance, security and management integration with no requirement to introduce additional platforms to support the required architecture
- Service Virtualization - performance and reliability, standards support for governance automation (UDDIv3, WS-MEX), standards-based closed-loop governance system
- Trust and Management Mediation - Interoperability across disparate partners and platforms, trust enablement and trust mediation complementing threat prevention systems
- Continuous Compliance and Validation - consistent policy implementation and enforcement across all stages of the lifecycle, preserving the fidelity of the governance models, structures and mechanisms
- Change Impact Mitigation - provides change management and impact analysis processes integrated with the governance workflow to ensure that changes to services or other assets don't cause major outages
- Consumer Contract Provisioning - provides offer, request, negotiation and approval workflows for service access, capacity, SLA and policy contracts

SOA.com offers governance tools for all leading SOA platforms: Microsoft (BizTalk), SAP (Net Weaver), IBM (WebSphere), Oracle (SOA Suite) and RedHat (JBoss). The SoA.com product suite consists of SOA Software's Portfolio Manager™, Repository Manager™, Policy Manager™, and Service Manager™ and forms a comprehensive Integrated SOA Governance Automation solution, with SOLA™ providing a governable Mainframe SOA platform.

Petals Master is an open source SOA Governance solution[18] that comprises a registry repository, the organization management tools including people and role management, and integrates with the service runtime environment (Petals ESB).

Software AG: According to a Gartner survey Software AG is raked as No 1 in suppliers for SOA governance technologies. The Software AG technology platform for SOA Governance provides tools from development-time lifecycle management to runtime service access mediation, monitoring, and management. The flagship product is called CentraSite™[19] and its main features are:

- Metadata Registry & Repository provides a unified registry and repository, a central, platform-independent store for defining and describing assets, and a catalogue of services, processes and related assets, such as XML schemas and business rules.

[18] http://petalsmaster.ow2.org/index.html

[19] http://www.softwareag.com/corporate/products/wm/soagovernance/centrasite/overview/default.asp

- Active Lifecycles tracks and guide the evolution of every service and process asset, from conception through retirement.
- Active Policies for automated lifecycle processes. An Active Policy defines a series of actions that are associated with an event, such as the addition, modification or lifecycle status change of an asset. When the event occurs, CentraSite executes the actions prescribed in the policy.
- Development-time policies support automated enforcement of policies along an asset's lifecycle at major governance checkpoints, such as funding approval, architecture and design reviews and operational readiness reviews as well as preloaded development-time policies, including metadata validation, WS-I compliance, approval workflow, permission provisioning, notifications e.g. are available
- Run-time policies: CentraSite also provides a powerful roster of policies that can be enforced when services are invoked at run-time to ensure correct and authorized use of services. Run-time policies include consumer authorization, message-level encryption and signatures and schema validation. Run-time policies are managed in CentraSite and enforced using webMethods Mediator.
- Smart policy provisioning: Once a policy is defined and approved, it is automatically applied to all assets that meet a specific criterion (defined with the policy).
- Custom policies: In addition to using CentraSite's pre-built policy templates, you can write custom policies in Java or Groovy scripting language. Those custom policies will have full access to all metadata available in CentraSite.
- Relationship Tracking manages the complex interdependencies among services, processes and IT applications using CentraSite's relationship tracking capabilities. Importantly, CentraSite also captures the relationships of services and processes with the people and organizations that built, operate and maintain them. CentraSite identifies and tracks many relationships automatically as services and processes are created, deployed and used. As needed, project teams can easily add new relationship definitions.
- webMethods Mediator is a service intermediary that enforces the run-time policies created in CentraSite. Mediator also virtualizes shared services, making it easy to change SOA. Combined with CentraSite, Mediator provides end-to-end governance of all services from development to run-time.

IBM has published its SOA governance position in a Redbook [202]. It also defines SOA Governance lifecycle that consist of four phases: Plan, Define, Enable and Measure. For governance policies IBM uses a policy lifecycle model containing the phases "Author", "Transform", "Enforce" and "Monitor". At the tool support side Tivoli Security Policy Manager and WebSphere DataPower SOA Appliance solutions push or pull artefacts so that policies associated with services can be interrogated at run time for compliance.

The Oracle SOA Governance Suite consists of four parts: Oracle Enterprise Repository, Oracle Service Registry, Oracle Enterprise Manager 11g and the Oracle Web Services Manager. The Web Service Manager a solution for policy management and security of service infrastructure. It provides visibility and control of the policies through a centralized administration interface offered by Oracle Enterprise Manager.

5.4 Deployment technologies

Software deployment can informally be defined as "all the activities that make a software system available for use " [203]. In particular there are the following groups of activities related to a typical deployment process (as described in [203]):

- Release – packaging of the software for the purpose of the deployment process
- Install – is the transfer of the software to a remote node and the associated configuration activities
- Activate – starting up the installed software
- Deactivate – stopping the running software
- Update – a special case of installation when a new version replaces the old one
- Adapt – modifying a software system that has previously been installed
- Deinstall – removing the software from a remote node
- Derelease – the end of a system life cycle when the system is withdrawn and development is stopped

Deployment activities can also be identified in most current software systems. [204] present another overview of stages of the deployment process, from the perspective of component-based applications. According to their work the process of deployment and configuration of component-based applications consists of the following steps:

- the mapping of the application's topology in terms of interconnected component instances onto the runtime environment's network of computing nodes,
- the installation of component implementations,
- the creation of component instances and
- the set-up of connections as well as the configuration of component properties.

Initially, service deployment technologies were based on manually edited scripts or configuration files, however this approach is error prone in case of complex modern systems. [205, 206] classify existing approaches to deployment into the following categories:

- Manual – simple to modify (just a change in natural language description of the process), good for simple systems which rarely change.
- Script-based – modifications carried out in configuration files, good for simple systems which can with frequent changes.
- Language-based – modifications through changes in a declarative language, good for complex systems with frequent changes.

Model-based - modifications through changes in models and policies, good for complex systems with frequent changes, better than language based due to better expressivity easier, maintenance and reuse possibilities.

An interesting measure proposed by [206] for the comparison of deployment solutions is Quality of Manageability (QoM), which can be described via quantitative measures such as: number of lines of code written for deployment; number of steps involved to deploy; lines of code to express configuration changes; time to develop, deploy, and make a change. QoM can also be described qualitatively by: ability to automate the management process, including adaptability to changes (e.g., failures,

load); robustness, expressed in terms of misconfigurations; expressiveness of management (e.g., ability to express constraints, dependencies, and models); barrier to first use of the deployment tool.

The deployment process largely depends on the environment used for hosting the running software, however typical systems used for development of distributed applications (such as CORBA [207], Enterprise Java Beans and Microsoft .NET) did not offer much support for automated deployment. To facilitate the deployment process of distributed applications, various middleware solutions such as [208, 209, 210] have been developed. The purpose of these solutions was to support the development and deployment process of distributed applications; however neither of them delivers a fully automated deployment environment.

The need for an automated deployment process becomes especially evident in case of large component based systems, and especially in case of Model Driven Architectures [211]. An initial solution has been proposed to the problem of deployment in the "Deployment and Configuration of Component-based Distributed Applications Specification" [212], but some researchers find this approach to be not sufficient. For example [213] proposes a generic architecture for a deployment framework capable of handling most of existing component technologies with an additional assumption that this environment should not force any changes to these underlying technologies.

In the area of web services automated deployment is perceived not as a mechanism for facilitating the development process, but rather as means to ensure better runtime qualities of the services. For example, an automated deployment mechanism can be used to redeploy a running web service to additional machines as required in order to provide dynamic scalability of the system [214], or in other words to support a desired level of offered QoS. In prior works related to distributed systems the concept of an automated redeployment mechanism was used to protect the system from network link failures [215].

[203] identifies the following challenging issues of a deployment process: Change Management for Installed and Running Systems, Dependencies among Components, Coordination, Large-Scale Content delivery, Managing heterogeneous platforms, Deployment Process Changeability, Integration with the Internet, Security: Privacy, Authentication, and Integrity. During the 13 years that have elapsed since this publication, researchers have managed to resolve most of these issues. However the dynamic evolution of software development technologies has introduced additional challenges for the deployment process.

One such problem is the predictability of the deployment process, which is especially important in the area of real-time distributed and embedded systems. In some cases, deployment order inversion can occur, causing software with lower priority (low-criticality operational string) to be deployed before software with high priority (high-criticality operational string) [216], which could affect the QoS of such a system.

Another problem arises from the existence of multiple component technologies that could be used together to deliver a single system. The deployment environment

must be flexible enough to support these various systems, and in particular (as stated in [210]) it should:

- Provide unification (ability to support multiple component technologies)
- Support deploying of heterogeneous applications
- Be driven by models

Ideally, deployment concerns should be separated from the underlying component technology. [204] compare three different approaches to component deployment and configuration: ITU-eODL, UML 2.0 and the Deployment and Configuration Specification of the OMG [217]. Each of these three approaches provides a solution for the deployment process, however the authors conclude that each of them focuses on different aspects of the process, and in the end do not provide an indication of which of these could be used as a basis for building a generic deployment solution.

In the remaining part of this section we will survey existing deployment technologies.

J2EE is an environment where deployment is a strongly integrated part of the development process. The deployment process is described by deployment descriptors, which "describe the contents of deployment units and configure components and applications to their environment. They also externalize the relationships between components, so those relationships can be managed without writing or changing program code" [Java 2002]. J2EE defines [218] five types of deployment descriptors: EJB deployment descriptors, Web deployment descriptors, Application and application client deployment descriptors, Resource adapter deployment descriptors for Java Connectors. The J2EE standard for application deployment is packaging them into WAR files, where the web.xml file configures various aspects of the application configuration.

The J2EE environment has a growing collection of frameworks and tools that facilitate application development, some of which are geared toward providing deployment support. These include: Ant[20] scripts that can automate multiple tasks including deployment, the CARGO[21] library which can be used to deploy applications to containers from ant or maven builds or the PLAY[22] framework that can easily handle deployment of applications into most popular application servers.

J2EE capabilities for development of wide area networks can be enhanced by applying additional platforms such as SNAP – Structured overlay Networks Application platform [219]. The idea behind SNAP is to deploy web applications on structured P2P networks for scalability. Among services offered by SNAP to application developers, the following are related to the issue of deployment: adaptation (realized as an automated process of preparing applications for deployment), deployment (the process of uploading a web application into the SNAP platform, secured with the use of administrator signatures), application location (SNAP tracks the location of each application in the P2P networks and redirects requests accordingly), activation on

[20] http://ant.apache.org/

[21] http://cargo.codehaus.org/

[22] http://www.playframework.org/

demand (services can be deployed and started upon requests from users if they are not running on any other machine).

Similarly multiple techniques exist for the deployment of web services. For example the Apache Axis[23] engine provides the following methods to deploy a web service:

- JWS Files – a Java source file copied to the server will be automatically compiled, and the methods offered by the class will be automatically associated with SOAP messages
- Using Web Service Deployment Descriptors (WSSD) - the WSSD file is capable fully configuring a deployed web service (e.g. by defining handlers for requests).

However the development of Apache Axis has been discontinued as it is replaced by Axis2[24]. One particular feature of Axis 2 is the so called 'hot deployment' which makes it possible to deploy a new service while the whole system is running. Similarly a 'hot update' offered by Axis 2 makes it possible to make changes to a web service without the necessity of restarting the system. The deployment of services into Axis2 is simply a matter of preparing a proper XML file that describes the web service, creating an appropriate package which includes the XML file, and copying that package to the Axis2 engine repository.

An interesting approach to web service deployment is the Automated Deployment Planner [220] for compositions of web services. This planner tool uses the Reo [221] coordination model, where Reo is used to carry out web service composition. The task of the planner tool is to generate a specification of a deployment plan based on available web-services specification and Reo circuits. The deployment plan specifies the target destination (resource) to which each web service should be deployed. Such deployment planning can be viewed as an optimization problem, and the authors propose to use a graph-based approach to obtain optimal planning results.

[23] http://ws.apache.org/axis

[24] http://axis.apache.org/axis2/java/core/

6 Monitoring

6.1 Quality of Service

In order to be able to monitor and adapt Services to the changes in a dynamic environment there is a need to specify requirements that will need to be fulfilled. The description of the requirements should be both understandable by human users (business intelligence expert) and machines (parsing, transformation). What needs to be monitored are functional and non-functional characteristics of the system which describe not only what and how the SOA-based system is performing but also in what conditions and on what performance level. The Quality of Service (QoS) of advanced communication services and applications needs to be defined precisely and also it is necessary to verify that QoS statements hold. For both purposes some means of QoS languages and models are required [222].

QoS is a generic term that describes different aspects of the systems and provides a way to measure, qualitatively, how a system is providing its expected services. When dealing with systems based on Information Technologies (IT), the QoS perceived by their users is a function of the QoS of all the building blocks of that system: network QoS, systems QoS, application QoS, etc. All these types of QoS are interrelated and depend on each other [223].

QoS management is critical for distributed service-oriented architectures because different clients often have different QoS requirements and each server must serve many concurrent clients with limited resources and ever-changing business rules [224].

[225] specifies QoS as a combination of metrics and policies. QoS metrics are used to specify performance parameters, security requirements and the relative importance of the work in the system. We define three types of QoS performance parameters: Timeliness, Precision, and Accuracy. QoS policies capture application-specific policies that govern how the resource manager treats an application: Examples of such policies are management policies and the levels of service. The translation between the two delay parameters is fairly simple; the translation between the two jitter parameters is not. A QoS based system should be able to dynamically adjust the amount of work performed (e.g., by using hierarchical encoding of video data, or resizing the video frame). This would allow the system to make trade-offs between the various QoS parameters, when sufficient resources are not available. [226] has been among the first to recognize the need for an integrated approach to resource management. They presented an integrated framework that deals with end-to-end application QoS requirements. The notion of flow is introduced as an important abstraction within the framework. On the other hand the notion of QoS on different abstraction levels had to be taken into consideration. [227] proposed using multi-level QoS models as a support for QoS specification in Multimedia Applications. It introduced QoS session profile which was later used for negotiation translation (QoS mapping) with a use of stream descriptor. Each of the descriptors corresponds to a media (e.g. H261 video), a set of media (e.g. MPEG2 audio&video system transport), or a media component

(e.g. hierarchical MPEG2 video Base Layer) and will result into an independent connection in the network.

In 1998 [228] presented a general Quality of Service Modelling Language (QML) in which it was possible to define multi-category QoS Specifications for components in distributed objects systems. QoS specification in QML facilitates the static decomposition of a software system into components with precisely specified QoS boundaries and facilitates dynamic QoS functions such as negotiations, monitoring and adaptation.

[229] proposed the object model QoSPath for specifying QoS in an adaptive QoS system. The model is aimed at letting application programs specify their QoS preferences to the system. Adaptive QoS Systems are attractive for multimedia services that are accessed via the Internet or mobile computers. An important feature of an adaptive QoS system is that it allows application programs to specify the QoS level desired, and the system to set the level according to the resources available. With an adaptive QoS system, the user should specify not only the target QoS, but also a QoS range to minimize the quality degradation resulting from resource shortages. A different approach was presented by [230] which focused on addressing QoS provisioning issues that are: specification, establishment and feedback. In the paper, authors applied a user-provider model to match at the interface level the needed QoS by user entities to the QoS capabilities offered by the provider. The model presented was based on the ISO QoS Framework [231].

In 2002 [232] presented a conceptual object model for specifying Quality of Service (QoS) that forms a basis for a UML profile for QoS. The conceptual model is based on CQML, a lexical language for QoS specification. The main goal of this work was to contribute to the OMG UML Profile Standard. The UML profile presented does not propose any graphical notation. Authors claimed that one of the most reasonable possibilities is to define QoS characteristics as tags that can be exploited in tagged values on the relevant modelling elements

Opposite to most approaches where QoS is handled in a discrete way, [233] introduced fuzzy-control approach for quality of service (QoS) adaptation, needed in distributed multimedia applications. The proposed approach uses a rule-based model that is used in the quality degree function, which allows mapping various application QoS parameters into a uniform metric, called quality degree, representing user's preference. For the same type of multimedia systems [234] proposed to use the Unified Modeling Language (UML) for QoS specification. The reason behind this approach was to provide an open, distributed processing reference model and focus on the computational viewpoint. To specify the QoS aspects of computational objects in UML, authors used a real-time logic called QL. The purpose of this UML-based model was to act as a template via which specific distributed system designs could be constructed. In the same year [235] presented the first standard initiative in order to provide a platform independent modelling approach for component-based system design.

QoS involves a multitude of properties beyond the application-specific aspects, including performance characteristics, availability, responsiveness, dependability, security, and adaptivity. In general, QoS specifications [236]:

- should allow for descriptions of quantitative QoS parameters (for example, jitter, delay, and bandwidth) and qualitative QoS parameters, such as central processing unit (CPU) scheduling policy and error recovery mechanisms, as well as adaptation rules;
- must be declarative in nature—that is, to specify only what is required but not how the requirement should be implemented; and
- need to be accompanied by a compilation process that maps the QoS specification to underlying system mechanisms and policies.

In 2004 [236] evaluated QoS specification languages according to the five criteria:

- Expressiveness: A good QoS specification must be capable of specifying a wide variety of services, their required resources, and corresponding adaptation rules.
- Declarativity: A QoS specification should be declarative in nature, so that applications need not cope with complex resource management mechanisms for ensuring QoS guarantees.
- Independence: Specifications should be developed independently from the functional code for readability and ease of development and maintenance purposes. Independence also lets developers associate a single application, at the user's request, with different QoS specifications.
- Extensibility: This criterion lets developers evaluate how easy a language can be extended for specifying new QoS dimensions, such as security, availability, and responsiveness.
- Reusability: Reusability is important when QoS specifications become large, because a new specification might just be an existing one with only minor refinements. Ideally, a QoS language, like other programming languages, should have reusable features. One condition necessary to help achieve reusability

The result of the evaluation is shown in the table:

QoS language	Expressiveness	Declarativity	Independence	Extensibility	Reusability
SafeTcl	Poor	Fair	Good	Poor	Poor
QoS-A	Good	Good	Good	Good	Poor
QuAL	Good	Poor	Poor	Good	Poor
Fuzzy control	Fair	Good	Good	Fair	Poor
HQML	Good	Good	Good	Good	Poor
QDL	Good	Good	Good	Good	Poor
QML	Fair	Good	Good	Good	Good

In 2004 two related approaches for developing QoS-aware Component-based applications using MDA were presented. First, one of them was presented in [237]. The authors presented an approach for end-to-end QoS requirements modelling based on UML. The model was presented as a starting point to derive platform-specific QoS requirements that might be applied and instrumented on clients, middleware, operator's performance management systems, service interfaces and server-side component infrastructure. The second one described in [223] introduced the Extended MDA modelling framework where different QoS-related UML profiles are used to model QoS requirements, QoS monitoring mechanisms and instrumentation mechanisms. It was based on a generative modelling technique to evolve from platform-independent models to platform-specific models, in which functional and non-

functional (QoS) contracts of component-based applications are jointly specified. The modelling has been transformed into different Enterprise Java Beans (EJB) components whose QoS can be monitored using a management system based on JMX (Java Management eXtensions) – Java based technology.

[238] presented a solution that was to provide a model that was generic enough for reuse across multiple domains in SOA based systems. The approach was named QoSOnt and presented a QoS Ontology for Service-Centric Systems. Authors realized that systems based on SOA have much in common with component-based systems. Moreover they took care of the business side of QoS management describing that in the situation where a service marketplace exists, customers, making the judgment of service quality a key issue, will trade quality against cost. To have confidence in services, clients will require service monitoring, service negotiation, and the formation of legally binding documents (which could take the form of service level agreements or service usage contracts). In its simplest form an ontology is simply a taxonomy of domain terms. However, taxonomies are of little use in machine reasoning. The term ontology also implies the modelling of domain rules. It is these that provide an extra level of machine "understanding".

Another approach that takes into account the business side of QoS management was presented in [224]. Authors stated that business rules often change from time to time so that businesses can adapt to the dynamic and competitive environment. They also claimed that a traditional constrained optimization approach is difficult to apply here, since system configuration and resource environment are dynamic, while accurate and complete models of different types of resources and their interactions with the environment are very difficult to build. They proposed an XML-based policy-based approach for specifying and enforcing QoS management in the distributed SOA. In this case a policy is a decision tree that can be described by a set of rules (not vice versa), but the tree imposes an explicit structure on these rules. This hierarchical structure of rules can be used, according to the divide-and-conquer philosophy, to mimic human's natural process of hierarchical decision-making. In the conclusion authors described that a generic policy language is designed for both admission control and resource management. Based on a decision tree, the language can be easily understood by policy administrators and efficiently implemented with simple compilation techniques.

The paper that introduces a specific framework for QoS management model is [239]. Authors present a QoS model specific for multimedia systems that supports dynamic adaptation by defining active architectural components to verify whether the system QoS parameters are in agreement with the application requirements. The model explores the concept of QoS ranges which parameters are defined as range of values. If any situation is detected in which the QoS negotiated is not in accordance, an adaptation process is initiated, observing the requirements established in the application description.

There is another interesting approach for QoS Specification based on ontologies that is described in [240]. The paper proposes a semantic approach to providing QoS parameter transparency in network service negotiations. Similar approaches in the network systems and Web Services areas are considered, and a new model is pro-

posed. The proposal tries not only to be more general, but also allows the comparison of specifications in the context of ontology inference. The ontology proposed acts as a base for a semantic-based approach for network service negotiation.

6.2 Service Level Agreement

QoS experienced by end-users results directly from the QoS offered by multiple service providers. Very often one service provider relies directly on another, and therefore the QoS of services offered by the first provider depends to some extent on the QoS of services offered by the other provider. A so-called Service Level Agreement (SLA) is a method for a service consumer to specify the desired level of QoS for a service he is paying for. An SLA is often a part of the contract between the service consumer and provider, and "it specifies the responsibilities of each party and the level of the service" [241].

According to the SLA model adapted by the SLAng language [242], SLAs can be classified into two main groups: inter-service (to specify QoS level offered by services) and intra-service (related to QoS of software components). Additionally SLAs can be classified into vertical and horizontal. Vertical SLAs relate to cases when a service provides infrastructure support for a client, while horizontal SLAs relate to cases when services work at the same level. Each of these two groups can be further divided into three types of SLAs as follows:

Vertical:

- Hosting (between service provider and host)
- Persistence (between a host and storage service provider)
- Communication (between application or host and Internet service providers).

Horizontal:

- ASP (between an application or service and ASP)
- Container (between container providers)
- Networking (between network providers)

This example shows that an SLA does not necessarily have to mean a contract between two parties, however the most typical usage for an SLA is in fact to specify an agreement between a service provider and a service consumer. Once an SLA is established between two parties, they both are bound by the rules set in that document. In particular, the service provider is expected to provide QoS parameters better than certain thresholds specified in the SLA. A violation of these thresholds, i.e. the degradation of the offered QoS, is usually associated with penalties such as for example a reduction in the service fee payable by the consumer. Therefore it becomes a necessity to accurately monitor QoS parameters of the offered service in order to detect SLA violations.

There are a number of purposes for monitoring QoS. First, the service consumer can use the monitoring data to detect violations of the SLA (and claim financial benefits). Second, the service provider can use monitoring data to prevent SLA violations from taking place (by adapting offered services to changing conditions). Third, so called 'intra-service' QoS control [242] can take place between components of running ser-

vices in order to perform runtime reconfiguration of software parameters when required to meet the desired QoS output level.

From the point of view of a service provider meeting the QoS conditions specified in an SLA is often a matter of optimization [243]. Lowering the level of QoS can often lead to a decrease in costs, while at the same time the SLA requires the QoS to be kept above a specified threshold. Based on available monitoring data the service provider performs adaptation of offered services to meet the QoS-demands of associated SLAs. The simplest approach to adaptation is to use collected monitoring data to improve the system design and implementation in order to deploy a new version of the service in the future. This approach however does not prevent SLA violations as it is in general carried out 'post-factum'. A more advanced solution capable of preventing SLA violations before they take place is automated service adaptation carried out during runtime. An important feature of such automated adaptation is the capability of the system to predict the occurrence of SLA violations [244]. Limiting the number of occurring violations through dynamic adaptation is referred to as SLA protection [243].

The requirements for the target QoS level specified in SLAs are called Service Level Objectives (SLO). In other words SLOs are "rules which specify the quality of service as numerical target values (plus associated penalties)" [244]. Each SLA defines a number of SLA metrics, which are then used to monitor the fulfilment of defined SLOs. For typical network-based services an SLO could be a statement that "the service provider must ensure that the system is available for at least 99% of the requests for each calendar day" [245]. This SLO relates to service availability, which is described through the 'daily percentage of handled requests' metric. The choice of metrics used when specifying SLOs is critical for a precise interpretation of the SLA, which in turn is a prerequisite for detecting SLA violations.

No	Description	Object	Unit
1	Availability	Hardware	Time hour, percent
2	Maximum down-time	Hardware	Hours or percent
3	Failure frequency	Hardware	Number
4	Response time	Hardware	Duration in minutes/seconds
5	Periods of operation	Hardware	Time
6	Service times	Hardware	Time
7	Accessibility in case of problems	Hardware	Yes/no
8	Backup	Hardware	Time
9	Processor time	Hardware	Seconds
10	Instructions per second	Hardware	Number per second
11	Number of workstations	Hardware	Number

Figure 12: hardware performance metrics as shown in [246].

A multidimensional categorization of SLA metrics for IT services has been proposed in [246]. The categories used to classify the metrics are: the service objects considered by the metric, ITIL processes in which the objects are involved, and the measurability of the metric. The identified service objects are: hardware, software, network, storage, service desk, and for each such object the authors list a number of typical metrics (as for example in Figure 12). Examples of listed ITIL processes include configuration management, problem management or incident management. Perhaps

the most important issue when choosing a metric for the SLA is the possibility for automated measurements of the metric during runtime monitoring.

A common pitfall of typical SLAs is that they are not formalized, but written in a plain text format. Without a formalized model for an SLA automated monitoring will not be possible. A formal SLA representation needs to be able to specify SLA metrics and the desired levels for these attributes (in [243] these are called Key Performance Indicators). Each metric should additionally be associated with a formally defined penalty so that an automated SLA violation monitoring system will be able to perform autonomous decisions regarding system adaptation. A number of formalized approaches to SLA specification can be found in the literature, and we will briefly describe them in the following part of this section.

IBM has proposed the Web Service Level Agreement (WSLA) framework [247,248] provides formalized tools and methodologies for defining and monitoring SLAs. The framework is oriented towards Web Services, but it can be applied to other inter-domain management scenarios as well. The framework utilizes an XML Schema-based language that makes it possible for service consumers and providers to unambiguously define variable SLAs. The WSLA language specification [249] specifies three sections for a typical SLA structure: parties (to identify all contractual parties), service description (specifies the characteristics of the service and its observable parameters and metrics), and obligations (defines service level objectives and action guarantees). The structure of the WSLA language is easily extensible and can be enhanced with domain or technology specific elements.

The Web Service Offerings Language (WSOL) [250] is an XML language based on an XML Schema, which is compatible with the WSDL standard. In terms of WSOL a web service offering is an SLA between Web Services) [251]. The main use of WSOL is to define multiple classes of a single web service through parameters such as functional constraints, some QoS constraints, simple access rights, price, and relationships with other service offerings of the same Web Service. Using the WSOL description of a web service enables selection of a more appropriate Web Service and service offering for particular circumstances.

The Rule-based Service Level Agreement (RBSLA) [252] language is a declarative language based on RuleML. The RBSLA language has been developed explicitly to provide means for specifying SLAs in a machine-readable syntax. Rules written in the RBSLA language are also formulated using the XML syntax, however they can be "translated into an underlying logical system and executed by a rule engine" [252]. The advantage of RBSLA language over other solutions is that it is based on an underlying logic system that allows the rule engine to make use of reasoning. Also, when RBSLA rules are translated into a logic program multiple simplifications can be applied to optimize the resulting code.

"The objective of the WS-Agreement specification [253] is to define a language and a protocol for advertising the capabilities of service providers and creating agreements based on creational offers, and for monitoring agreement compliance at runtime". The specified approach in itself is supposed to be general to allow multiple usage scenarios. The basic concept is that the service provider and consumer exchange messages according to a predefined protocol, in order to establish at runtime rules

regarding their usage of the service. The agreement is represented as an XML document that consists of the following top-level elements: Agreement, AgreementId, Name, Term, AgreementContext. The WS-Agreement specification defines further the allowed nested tags for each of these elements.

The Web Service Modeling Ontology (WSMO) [254] provides a framework and a language for describing multiple aspects of web services. The basic elements of WSMO are: ontologies (which provide required terminology), web service descriptions (related to the functional and behavioural aspects), goals (users desires), mediators (to automatically handle interoperability issues). WSMO as in itself is not focused nor designed for specifying SLAs, however it has been extended [255] with the ability of modelling QoS characteristics of services, which could be used for constructing SLAs. The interesting feature of WSMO is that it uses an ontology and therefore provides a very high level of semantics to the descriptions of web services.

The SLAng language [242, 256, 257, 258] has been developed for formally describing SLAs for the purpose of their monitoring. The main features of the SLAng language is that it defines the SLA vocabulary, its structure is derived from industrial requirements and the meaning of SLAng is defined with reference to a model of service usage. The SLAng language is specified using UML (as a meta-model), the syntax of SLAng is defined through an XML schema, while the constraints defined in SLAs are expressed using the Object Constraint Language (OCL). The semantics behind OCL is important as it reduces the ambiguity when interpreting the meaning of an SLA. SLAs are modelled as a sequence of actions, where each action has some associated events (i.e. an action can give rise to any number of associated events). SLAng can be used to construct systems with reliable QoS characteristics, and can also be used as basis for implementing violation monitoring.

An interesting approach [259], much different from all of the presented above, is based on using performance trees for SLA metric specification. Performance trees were developed for the graphical specification of complex performance queries. A performance tree query is represented as a tree structure where nodes can be either operations or values. The authors of this work have developed an alternative query building mechanism called Natural Language Query Builder. These tools can be used to provide an easy to use way to specify SLA metrics. The same tools can later be used to monitor compliance of a system with the SLA.

6.3 Runtime monitoring

Systems based on SOA assumes loose coupling of services, which are accessible over the network so they can be easily reused to develop new applications. Examples of such systems are Web services, which can be seen as functional building blocks accessible over standard Internet protocols independent of platforms and programming languages. With the time, systems based on SOA are getting more and more complex and thus they become more prone to errors. Even if the given service is fault-free during testing phase, there is no guarantee that it will work properly during run-time. Moreover, the content of the services can be modified by its provider without prior notification; hence unaware user of the service may see these changes as an error.

In order to provide a control over such complex systems, runtime monitoring is required. A proper runtime monitoring needs to measure different system parameters or available resources based on two activities [260]:

- Detection of the prior unknown properties of the system - it is done by collecting information about the given system, creating a proper model of such system, and designing parameters of such model.
- Verification of the defined properties of the system – it is done by comparing the actual state of the system with the assumptions about the system at the given time. For example, it checks if the system fulfils the restrictions defined by QoS policies.

Several approaches of run-time monitoring were described in the literature. However, from the architecture point of view different aspects should be taking into account, such as features being monitored or the monitoring model that is used. Implementation of the monitoring architecture should also focus on the communications mechanisms, protocols and session management. In [261] authors propose to classify existing approaches of monitors with respect to the design of monitoring architecture into 2 logical groups:

- Heavy – weight monitor - a single monitor supports the monitoring of different aspects of a Web service.
- Light - weight monitor - a single monitor supports the monitoring of one aspect of a Web service

Based on this classification, authors proposed a theoretical framework for run-time verification of conversational Web services, which aims to provide a holistic monitoring framework enabling the integration of different verification tools. As a result, an extensible architecture has been proposed, which supports the integration with the existing service-oriented architectures and allows the use of different monitoring approaches for message interception and session handling. Moreover, message interception approaches have been classified as:

- handler-based: a handler is attached to the monitored service,
- wrapper-based: the monitored service is wrapped within another service,
- proxy-based: an intermediate node acts as a network proxy.

Another example of the business process monitoring classification from the architecture point of view can be divided in two groups [262]: Server side and Client side monitoring.

Server side monitoring is very accurate because it monitors service's parameters of interests during service operation, such as QoS attributes. The drawback of this technique is that it requires access to the current implementation of the service, which is not always available for outsiders. One example of such monitoring is presented in [263], which is used for developing adaptive service-based software systems (ASBS). The main idea is to develop a performance model through a series of controlled experiments, and then to design QoS monitoring and adaptation modules based on the obtained performance models. The developed modules are the part of the service implementation. The Windows Communication Foundation (WCF) is another platform that allows monitoring service's performance by incorporating in the

implementation of the service Performance Counters (WPC) (that can be checked at the run-time) [262, 264]

Client side monitoring is independent of service implementation but it does not always have access to the actual values of service's parameters as compared to server-side monitoring. Usually, it sends probe requests to the monitored service and based on received answers the values of the monitored parameters are estimated (sometimes inaccurately). Another problem of this technique is the overhead introduced by frequent probing. However, there are attempts to reduce this overhead (for example, by using Time Series Forecasting (TSF) [265]). One of the examples of client side monitoring is a policy-driven distributed framework for monitoring quality of web services [266]. The proposed monitor is user-centric, loosely coupled with service providers, and is integrated with QoS registry Q-Peer. Instead of sending probe requests, it observes the interaction messages in SOAP format, and calculates QoS metrics based on the information included in captured messages.

Runtime monitoring of business processes can be treated as a dynamic analysis of runtime events. Such events can be analyzed:

- online – during the execution of the application, or
- offline - after the execution has been terminated.

This way monitoring process is able to assess the quality of specific application. The main advantage of offline techniques is the possibility to examine the particular system in different ways using tools, which cannot be used during runtime. However, an offline technique requires storing large amount of collected data, which could be hard to manage and analyze. On the other hand, online techniques collect only the small amount of predefined system properties. This gives the ability for prompt reaction on interested events as well as detection of critical situation.

Depending on the types of the monitored properties, runtime monitoring of web services can be assigned into 2 groups:

- Global monitoring properties – analysis of orchestrated obligations [267, 268, 269, 270, 271];
- Local monitoring properties – events of single service;

Runtime monitoring is also used to measure different system parameters or available resources. In [272] the authors proposed to use runtime monitoring of conversations between services or business processes as a means of checking behavioural correctness of the entire Web service system. They identify a subset of UML 2.0 Sequence Diagrams as a property specification language and show that it is sufficiently expressive for capturing safety and liveness properties. The semantic of proposed subset of sequence diagrams have been transformed into automata to enable conformance checking of finite execution traces against the specification.

As an alternative, online runtime monitoring presented in [267] is performed by a framework capable of automatically associating business rules with relevant processes involved in a user request. This framework plans and monitors the execution of the request against services underlying these processes. For this purpose, XSAL assertion language has been proposed. Such language express business rules in the

form of assertions over business processes. However, such monitoring technique is restricted to local properties – events of a single service.

There are also other approaches that deal with monitoring of the assertions over service enabled business processes. One of them is WS-Policy framework [273], which provides a general-purpose model for describing a broad range of service requirements, preferences, and capabilities. Another example is RuleML [274], technique for expressing business rules over semantically annotated service.

In [268, 269, 270] proposed monitoring technique are used to check service pre- and post-conditions associated to external service invocations. Specifically, [268] presents approach for monitoring the execution of composed Web services. In this approach, dynamic compositions are represented as BPEL processes, which can be monitored at run-time to check whether individual services comply with their contracts. Authors presented two approaches: one based on late-binding and reflection and the other based on a standard assertion system. Extension of this idea was presented in [269] where runtime monitoring was extended to support dynamic monitoring of WS-BPEL. In [270] the authors proposed runtime monitoring technique based on model driven solution.

In the work [275, 276, 277] offline monitoring techniques are presented, which are able to handle global and local properties. The proposed framework uses BPEL4WS language for a service composition process description, assumes systems composed of web-services, and uses event calculus to specify the properties to be monitored.

The main classifications and its examples of run-time monitoring approaches have been described. However, several other approaches of monitoring framework of web services have been proposed in the literature, such as: validation of predefined interaction constraints using finite state automata [271], handlers to intercept messages and measure the responsiveness of a service [278], UML 2.0 Sequence Diagrams as a property specification language [272], monitoring by employing workflow graphs as the underlying specification language, for generating monitors that are exposed as Web services [279], data collection, including message interception for monitoring [280], event-based monitoring of process metrics across participants in a choreography [281].

In the INDENICA project we will focus on the definition of the generic architecture of monitoring event model. Such model should be able to all event from any existing service platforms. Moreover, we will investigate the possibility to develop monitoring engine detailed enough to create rules for triggering adaptation actions. Therefore, we would like to provide an event hierarchy, which serves as the main classifier for concrete events. Proposed hierarchy of monitoring event model will be evaluated in the real world scenario.

6.4 Infrastructure for runtime supervision

In general, runtime supervision has been mainly addressed from an architectural point of view. This means that adaptation is applied by simply changing the architecture of the system. Since traditional architecture-level approaches do not allow to change the adaptation policy at runtime, new research propose to use live models that inject adaptation strategies only when necessary, without hardcoding all adap-

tations at design-time. Finally other approaches focus their attention on service compositions, providing an infrastructure for monitoring and/or adapting compositions at runtime.

In the INDENICA project we will look for other flexible ways to provide runtime validation platform. One of the solutions that will be evaluated is a middleware which exploits Publish/Subscribe techniques to distribute information. For this purpose we will test and check different kind of event processing components such as ESPER. Moreover, the monitoring efficiency of choosen components will be evaluated. Implemented middleware will also provide pipe and filter architecture to easily connect services and dynamically process its data. The proposed monitoring architecture will also contain collections of probes which acquire required data. Such mechanism will be based on the push/pull middleware as well as periodically checking of data.

6.4.1 Architecture level self-adaptation

Traditionally, adaptation has been addressed at the architectural level, since this point of view allows the developer to shift the focus away from lines-of-code to coarse-grained components and their overall interconnection structure. Furthermore, as an abstract model, architecture can provide a global perspective of the system and expose important system-level behaviours and properties. These approaches can adopt either internal self-adaptation mechanisms [282] or external mechanisms [283] in a closed control loop.

Oreizy et al. [282] support software evolution by maintaining the architectural model of the system that is used as a basis for change. Adaptations are performed on the architecture and allowed changes comprehend adding/removing components and structural reconfiguration, i.e., recombining existing functionality to modify overall system behaviour. Changes to be performed are represented internally through an imperative language. It is also possible to enforce a particular policy that, for example, preserve specific component connectors or satisfies a set of invariants to preserve the integrity of the system. Conversely, Garlan et al. [283] propose an external framework, Rainbow, to monitor and adapt the system behaviour at runtime in a closed control loop. The infrastructure of Rainbow provides probes/gauges for data collection, and resource discovery mechanisms and effector mechanisms to carry out the actual system modifications. Probes collect basic system properties, while gauges aggregate the information coming from the probes to update the properties described in the architecture model. Obviously Rainbow keeps a translation repository to maintain the mapping between the runtime data and the properties of the architecture model. Furthermore, the concept of architectural style is enriched with a notion of adaptation style that is composed of operators and strategies. The former determine a set of style-specific actions that can be performed on a system element to alter its configuration. While the latter specify the adaptation that can be applied to move the system away from an undesirable state. This approach works under the assumption that any target system has access hooks for monitoring and adaptation, and a set of probes is always available to provide information about system properties.

The main problem is that these approaches do not provide an explicit support for the evolution of the system. In particular, they constrain the set of possible adaptations that can be performed without allowing one to change the adaptation policies as long as the system evolves. Flurey et al. [284] adopt models to specify the architectural variants of the systems that can be applied on the base system architecture. At runtime a valid application configuration is generated by composing the base and the variant model, depending on a set of adaptation rules and the current context of the executing system. The main limitation of this approach is that it is only able to handle foreseen adaptations: architectural variants are statically defined for a specific set of adaptation rules, a context and a predefined base architecture of the system. At runtime it is not possible to introduce new variants or different adaptation rules as the system evolves.

Other approaches propose goal-oriented middleware [285, 286] that are inspired from the three-layer layer framework already proposed by Kramer and Magee [287]. Sykes et al. [285] developed a framework that includes a component layer that is concerned with near real-time reactive behaviours, the change management layer that is concerned with the composition and sequencing of those behaviours, and the goal management layer that provides a system controller that prescribes the behaviour required to the system to satisfy its given functional goals. The authors focus on the middle layer that generates valid configurations with respect to the behavioural capabilities prescribed by the goal layer. It also checks which of the valid configurations conform to any structural constraint provided by the user (e.g., whether the configuration conforms to an architectural style). Finally, this layer takes into account a set of non-functional properties to make the best choice among configurations. However, one of the main drawbacks of this approach is that even if the system behaviour can change over time, the policy adopted by the goal layer cannot evolve.

PLASMA [286] is a three-layered framework that allows to automatically modifying the current system architecture in case it is inadequate to achieve a specific system goal. To this aim, PLASMA adopts application and adaptation domain models. The former captures the possible states of application components and the actions that these components can perform. The latter represents a set of architectural configurations (states) and actions that can be performed by the system to move from a state to another one. The input provided is the problem description (the initial architectural configuration and a goal), the description of the available components, expressed in ADL, and the components implementation. From the ADL model the application domain model describes a plan to achieve the goal specified as input. Then the target architecture topology required to run that plan is generated. The target architecture and the current architecture topology represent respectively the initial and target state of the adaptation problem and the necessary reconfiguration actions to move from the current state to the target one can be generated automatically. This approach has the main advantage to support the automatic generation of architectural-level adaptations, but it does not consider non-functional requirements in the selection of a suitable adaptation.

6.4.2 Runtime supervision of service compositions

This section describes some approaches that support the runtime supervision (monitoring and recovery) of service compositions. Supervision techniques mainly differ in the requirements they deem important: a monitoring approach can be more or less intrusive, more timely in discovering anomalies, or more tailored towards the analysis of functional properties instead. For example, we can mention Dynamo [288] and ALBERT [289]. Dynamo is a synchronous and assertion-based monitoring approach. It stops a process every time it interacts with a partner service, and checks any pre- or post-condition, expressed in a custom language (WSCoL). Despite Dynamo is very intrusive (it uses AOP [290] technology), it is very good at discovering anomalous behaviours as soon as they occur. On the other hand, ALBERT [289] is an asynchronous approach that verifies a set of temporal properties. In this case, all the assertions are checked in a separate thread, while the process is only stopped to collect the run-time data needed to check them. ALBERT is much less intrusive, but can capture anomalies only when the process had already proceeded beyond the point in which they were generated.

It is clear that both approaches have strong advantages, but also evident weaknesses. The same considerations can be argued for other approaches [275, 173]. Mahbub and Spanoudakis [275] propose a framework for the run-time validation of behavioural assumptions that are expressed in terms of event calculus. The approach requires that designers reason at a low level of abstraction, since they need to configure probes for system events, and express the properties they want to check. The approach is very unintrusive, and produces post-mortem results. It is suitable in situations when the designer is interested in keeping a low profile, and in fixing only future iterations of a process. Amongst other unintrusive approaches we consider VieDAME [173], where process-internal state is inferred in an unblocking way, by analyzing the incoming and outgoing messages the process exchanges with partner services. It accumulates data as the process instances are run, aggregating previous data to calculate QoS values such as response time, accuracy, or availability. Despite the intrusiveness is minimized, the approach does not consider external data coming from the environment. Furthermore, VieDAME concentrates on a fixed set of non-functional properties, without allowing the designer to define its own complex properties. Instead, we argue that flexibility is a key aspect and monitoring approach must support the validation of any kind of requirements, depending on the stakeholder's personal needs.

A lot of attention has also been given on assessing SLAs (Service Level Agreements). An SLA defines a contract between a set of customers and providers on particular QoS properties. The contract also devises a set of penalties that may be paid by one of the contractors if he/she violates any guarantee term. Keller and Ludwig [291] propose a framework to define and monitor SLAs, focusing on QoS properties such as performance and costs. In this case, measurements are performed by probing client invocations or retrieving metrics from internal resources. Sahai et al. [292] describe an automated and distributed SLA monitoring engine. The monitor acquires data from instrumented processes and —by analyzing the execution of activities and message passing— verifies the SLAs. Finally, Skene et al. [257] propose the SLAng language for SLAs, described using meta-modelling techniques derived from the

MDA toolset. All these approaches allow monitoring the properties defined on the negotiated contract, which are overall QoS properties that must be satisfied by the entire process (e.g., the overall cost, the overall response time, etc.). For this reason, these approaches neglect environmental properties that must be satisfied by the context. Conversely, our approach considers a set of requirements defined only at the service user side, i.e., at the BPEL process side. We aim at monitoring a wider set of properties, like functional and non-functional requirements or domain assumptions. In particular, we are convinced that monitoring context data is fundamental, since context variability can be one of the main reasons to perform adaptation.

Other works [293] propose a middleware to support the fault-tolerant execution of services based on standard WS-policies [294] (e.g., provisioning policy, clients requirements). The same authors also provide a language [295] to specify policies in terms of general goal assertions. This way, a policy can define the source of the monitoring data (both process and context data), the modality of the monitoring (synchronous or asynchronous), a set of supervision parameters, and the actual properties that must be satisfied.

Although these approaches offer several advantages, none of them has been able to provide a one-stop solution. Although each is particularly effective in its own subdomain, none provides a holistic solution that easily accommodates the very different needs of the clients —in terms of qualities of interest and required analyses. Starting from this assumption Comuzzi and Spanoudakis [296] propose a hierarchical framework to monitor properties at different layers of the service-based system such as service composition (the workflow), service invocation and execution (i.e., the set of resources on which each single service executes). It integrates different kinds of monitors designed to assess properties at different layers of a service-based system. Different monitors and event captors may be plugged in the proposed architecture as long as they are able to perform monitoring using events and the monitoring capabilities of the workflow and local services. This approach [297] also requires the assessment of the monitorability of an SLA, and the dynamic set-up of monitoring resources for checking an SLA. This work represents a big advance towards the integration of different monitoring components, but the integration of different recovery techniques has been still neglected.

Recovery of service compositions has received, in general, less attention from the research community. We are still far from having adequate solutions for issues such as dynamic instance/class re-configuration or process rollback (techniques already present in classical database, workflow, and web-based systems).

The run-time recovery of BPEL processes has been tackled by Dynamo [288], which provides a suitable language (WS-ReL) that offers a set of pre-defined atomic recovery actions, and constructs for combining them to build more complex recovery strategies. All atomic actions have process instance validity, meaning no recovery is performed on the process definition itself. The library of possible actions provides means to perform service and partnerlink substitution (even when the new service has a different interface), notifications (via email), calls to process event handlers, as well as simple rollbacks. Most approaches do not go beyond dynamic rebinding in BPEL processes, and most adopt proxy-based solutions. This is the case of Ardagna et

al. [170], who propose the PAWS (Processes and Adaptive Web Services) framework, and Colombo et al. [298], who propose SCENE. Moser et al. [173], on the other hand, provide dynamic substitution using an AOP-based extension of ActiveBPEL.

7 Conclusions

This document offered a wide summary of the main technologies, approaches and results available in the domain of service-based systems. Even if the main interest of INDENICA is in the concept of platform as a service, the report addresses the global domain of services and service-centric systems to provide the consortium with a common background, frame the solutions developed in the project, and also identify significant gaps and challenges that must be addressed to tackle the integration of service platforms.

To conclude there are different challenges a service platform and its design in the context of Software Product Line Engineering (SPLE) need to address. Here is a list of challenges without claiming to be complete:

- A design process is required that combines SOA-based design methods (such as IBM's SOMA or the ISE Methodology) with design of Software Product Lines.
- Requirements for service platforms must be suitably elicited and represented. A platform can be seen as the union of "all" the applications that could be implemented on top of it, but requirements at platform level, along with the identification of the variability that must be embedded in the platform, require special-purpose solutions.
- Adaptation is another key issue. It is a crosscutting concern, and it must be addressed from the very beginning. Adaptation must be modelled explicitly, it must be embedded in the platform at runtime, and it must be suitably probed to keep it under control.
- Semantic Service Descriptions are required that also contain "meta information" helpful for (semi-)automatic integration of SOA-based core assets into concrete but varying application contexts. Explanation: The service platform shall support easy integration of industrial protocols such as OPC/UA or BACnet. This is necessary to integrate lower layers of the automation pyramid. To ease the implementation of appropriate adaptation technologies such as SCA might be helpful. In addition, MDSD approaches could be leveraged. The same holds for industrial data and document formats that need to be integrated into a SOA platform. As an example we could consider a service that should have the same interface but different possible implementations, thus integrating into different embedded environments.
- The Service platform must support (health) monitoring and further crosscutting concerns that are essential for operation in industrial contexts. There must be a way to separate such crosscutting concerns and introduce them systematically into the platform. The results of the AMPLE Project[25] and THESEUS TEXO should be leveraged here.
- Configuration aspects, versioning aspects, and documentation aspects need to be supported by the service platform design to enable easy application development. Thus, best practices how to design and build the platform respective of

[25] http://www.ample-project.net/

these issues and how to provide an appropriate production plan need to be investigated.

These challenges are currently being addressed at the ERP layer, but the aforementioned existing approaches also need to be adapted to the lower (i.e. industrial) layers of the TIA Pyramid.

8 References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. Computer, 40(11):38–45, 2007.

[2] M.P. Papazoglou, Web Services: Principles and Technology, Prentice Hall, 2007.

[3] T. Erl; *SOA Design Patterns*; Prentice Hall; 2009

[4] World Wide Web Consortium. Web Services Activity. http://www.w3.org/2002/ws/

[5] The OSGi Alliance. OSGi Service Platform, Release 3. IOS Press, 2003

[6] P. Bellavista, A. Corradi, and C. Stefanelli. Mobile agent middleware for mobile computing, Computer, vol.34, no.3, pp.73-81, Mar 2001

[7] D. Zhang, X. Hang Wang, K. Hackbarth. OSGi based service infrastructure for context aware automotive telematics, 59th IEEE Vehicular Technology Conference, pp. 2957-2961 Vol.5, 2004

[8] E. Kaasinen. User needs for location-aware mobile services. Personal and Ubiquitous Computing 7(1), pp. 70-79, Springer, 2003.

[9] Organization for the Advancement of Structured Information Standards. Devices Profile for Web Services (DPWS). http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01

[10] H. Bohn, A. Bobek, and F. Golatowski. SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains. International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006.

[11] Organization for the Advancement of Structured Information Standards. UDDI Version 3.0.2. http://uddi.org/pubs/uddi_v3.htm, 2004.

[12] Organization for the Advancement of Structured Information Standards. Electronic Business using eXtensible Markup Language. http://www.ebxml.org, 2006.

[13] A. Michlmayr, F. Rosenberg, P. Leitner, S. Dustdar. End-to-End Support for QoS-Aware Service Selection, Invocation and Mediation in VRESCo. In IEEE Transactions on Services Computing (TSC), 2010

[14] International Business Machines Corporation (IBM). WebSphere Service Registry and Repository. http://www.ibm.com/software/integration/wsrr/, 2002.

[15] N. Kavantzas, "Web Services Choreography Description Language 1.0," editor's draft, 3 Apr. 2004, W3C; http://lists.w3.org/Archives/Public/www-archive/2004Apr/att-0004/cdl_v1-editors-apr03-2004-pdf.

[16] D. Scheibli, "Analysis of the State of the Art and Definition of Requirements", Deliverable D1.1 from 4Caast, available from http://4caast.morfeo-project.org/documentation

[17] Open SOA Consortium: http://www.osoa.org/display/Main/Service+Component+Architecture+Home

[18] David Chappell; *Introducing SCA*; 2007; http://www.davidchappell.com/articles/introducing_sca.pdf

[19] P. C. Clements, L. Northrop. Software Product Lines. Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley, November 2005.

[20] F. J van der Linden, E. Rommes, K. Schmid. Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, Springer, 2007.

[21] K. Schmid. Scoping Software Product Lines – An Analysis of an Emerging Technology, Proceedings of the First Software Product Line Conference (SPLC'1), Kluwer, pp. 513-532, 2000.

[22] L. Chen, M. A. Babar, N. Ali. Variability management in software product lines: A systematic review, In Proceedings of the 13th International Conference on Software Product Lines (SPLC), pp. 81 – 90, 2009.

[23] S. Robak. Feature Modeling Notations for System Families, International workshop on Software Variability Management (SVM), pp. 58 – 62, 2003.

[24] M. Sinnema, S. Deelstra. Classifying variability modeling techniques, Information and Software Technology, Vol. 49, pp. 717 – 739, 2007.

[25] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, Feature-oriented domain analysis (FODA) feasibility study, CMU/SEI-90-TR-21, SEI, CMU, Tech. Rep., 1990.

[26] K. Czarnecki, U. W. Eisenecker, Generative Programming – Methods, Tools, and Applications, Addison Wesley, 2000.

[27] K. Schmid, R. Rabiser, and P. Grünbacher, A Comparison of Decision Modeling Approaches in Product Lines, in Proceedings of the Fifth International Workshop on Variability Modelling of Software-intensive Systems (VAMOS'11), pp. 119 – 126, 2011.

[28] Y. Bontemps, P. Heymans, P.-Y. Schobbens, J.-C. Trigaux. Semantics of FODA Feature Diagrams. The Third Software Product Line Conference (SPLC04); Software Variability Management for Product Derivation - Towards Tool Support, 2004.

[29] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux. Feature Diagrams: A Survey and a Formal Semantics. Requirements Engineering, 14th IEEE International Conference, 2006.

[30] A. Classen, A. Hubaux, and P. Heymans, A formal semantics for multi-level staged configuration, in Proceedings of the Third International Workshop on Variability Modelling of Software-intensive Systems (VAMOS'09), 2009.

[31] K. Kang, S. Kim, J. Lee, K. Kim, E. Shin, M. Huh. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures, Annals of Software Engineering, Vol. 5, pp. 143 – 168, 1998.

[32] M. Griss, J. Favaro, M. dl Alessandro. Integrating Feature Modeling with the RSEB. In Proceedings of the Fifth International Conference on Software Reuse, pp. 76 – 85, 1998.

[33] M. Riebisch, K. Böllert, D. Streitferdt, I. Philippow. Extending Feature Diagrams with UML Multiplicities, Proceedings of the Sixth Conference on Integrated Design and Process Technology (IDPT 2002), 2002.

[34] P. van den Broek, I. Galvão, J. Noppen. Merging Feature Models, Proceedings of the First International Workshop on Formal Methods in Software Product Line Engineering (FMSPLE 2010) at (SPLC 2010), pp. 83 – 90, 2010.

[35] Reuse-Driven Software Processes Guidebook, Version 02.00.03, Software Productivity Consortium Services Corporation, Technical Report SPC-92019-CMC, November 1993.

[36] K. Schmid, I. John. A Customizable Approach to Full Lifecycle Variability Management, Science of Computer Programming, Vol. 53, No. 3, pp. 259-284, 2004.

[37] H. Gomaa. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures, Addison-Wesley, 2004.

[38] S. Azevedo, R. J. Machado, A. Bragança, H. Ribeiro. The UML «extend» Relationship as Support for Software Variability, Proceedings of 14th International Software Product Line Conference (SPLC 2010), pp. 471 – 475, 2010.

[39] K. Pohl, G. Böckle, F. van der Linden. Software Product Line Engineering Foundations, Principles, and Techniques, Springer, 2005.

[40] Y. Gil, S. Kremer-Davidson, I. Maman. Sans Constraints? Feature Diagrams vs. Feature Models, Proceedings of 14th International Software Product Line Conference (SPLC 2010), pp. 271 – 285, 2010.

[41] A. S. Karataş, H. Oğuztüzün, A. Doğru. Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains, Proceedings of 14th International Software Product Line Conference (SPLC 2010), pp. 286 – 299, 2010.

[42] J. Greenfield, K. Short, S. Cook, and S. Kent. Software Factories: Assembling Applications with Patterns, Frameworks, Models & Tools. J. Wiley and Sons Ltd., 2004.

[43] T. Stahl and M. Völter. Model-Driven Software Development: Technology, Engineering, Man- agement. Wiley, 2006.

[44] A. Brown. An Introduction to Model Driven Architecture - Part I: MDA and today's systems. http://www.ibm.com/developerworks/rational/library/3100.html, Feb. 17 2004.

[45] S. J. Mellor, A. N. Clark, and T. Futagami. Guest editors' introduction: Model-driven development. IEEE Software, 20(5):14–18, 2003.

[46] I. Kurtev, J. Bézivin, F. Jouault, and P. Valduriez. Model-based DSL frameworks. In OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pages 602–616, New York, NY, USA, 2006. ACM.

[47] OMG. Model-Driven Architecture. http://www.omg.org/mda, 2003.

[48] OMG. Meta Object Facility (MOF TM) 2.0. http://www.omg.org/spec/MOF/2.0/HTML, Jan. 2006.

[49] OMG. Unified Modelling Language (UML) 2.0. http://www.omg.org/spec/UML/2.0, July 2005.

[50] OMG. Object Constraint Language (OCL) 2.0. http://www.omg.org/spec/OCL/2.0, May 2006.

[51] A. van Deursen, P. Klint, and J. Visser. Domain-Specific Languages: An Annotated Bibliography. http://homepages.cwi.nl/~arie/papers/dslbib, Mar. 1998. (accessed 2009/02/18).

[52] L. A. Walton. Software Design For Reliability and Reuse. http://www.spatial.maine.edu/ ~lisa.walton/dsl.html, 1996.

[53] D. S. Wile and J. C. Ramming. Guest Editorial: Introduction to the Special Section "Domain-Specific Languages (DSL)". IEEE Trans. Software Eng., 25(3):289–290, 1999.

[54] U. Zdun. Concepts for Model-Driven Design and Evolution of Domain-Specific Languages. In International Workshop on Software Factories - OOPSLA 2005. Software Factories, 2005.

[55] D. Frankel. Model Driven Architecture: Applying MDA to Enterprise Computing. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[56] Klaus Pohl: Requirements Engineering. Fundamentals, Principles, and Techniques. Springer, Berlin, 2010; ISBN 978-3-642-12577-5

[57] Karl E. Wiegers: Software Requirements 2003. Microsoft Press, Redmond, Washington, 98052-6399

[58] J. Schmied, P.-R. Wentzel, M. Gerdom, U. Hehn: Mit CMMI Prozesse verbessern! (Umsetzungsstrategien am Beispiel Requirements Engineering). dpunkt-Verlag 2008; ISBN 978-3-89864-538-6

[59] I. Sommerville, P. Sawyer: Requirements Engineering: A good practice guide; John Wiley & Sons 1997; ISBN: 0 471 97444 7

[60] D. T. Ross and K. E. Schoman. Structured Analysis for Require- ments Definition. IEEE Transactions on Software Engineering, 3(1):6–15, 1977.

[61] John Mylopoulos, Lawrence Chung, and Brian Nixon. Representing and Using Nonfunctional Requirements: A Process- Oriented Approach. IEEE Transactions on Software Engineering, 18(6):488–497, 1992.

[62] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal- Directed Requirements Acquisition. Science of Computer Programming, 20(1-2): 3–50, 1993.

[63] Axel Van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In Proceedings of the 5th International Symposium on Requirements Engineering, pages 249–263. IEEE Computer Society, 2001.

[64] Alistair Sutcliffe. Scenario-Based Requirements Engineering. In Proceedings of the 11th International Conference on Requirements Engineering, pages 320–329. IEEE Computer Society, 2003.

[65] Eric Yu. Towards Modelling and Reasoning Support for Early- Phase Requirements Engineering. In Proceedings of the 3rd Inter- national Symposium on Requirements Engineering, pages 226–235. IEEE Computer Society, 1997.

[66] Ariel Fuxman, Lin Liu, John Mylopoulos, Marco Pistore, Marco Roveri, and Paolo Traverso. Specifying and analyzing early requirements in Tropos. RE, 9(2):132–150, 2004.

[67] Axel van Lamsweerde. Requirements Engineering: From System Goals to UML Models to Software Specifications. John Wiley, 2009.

[68] Ariel Fuxman, John Mylopoulos, Marco Pistore, and Paolo Traverso. Model Checking Early Requirements Specifications in Tropos. In 5th International Symposium on Requirements Engi- neering, pages 174–181. IEEE Computer Society, 2001.

[69] Fabiano Dalpiaz, Raian Ali, Yudistira Asnar, Volha Bryl, and Paolo Giorgini. Applying Tropos to Socio-Technical System Design and Runtime Configuration. In Proceedings of the Workshop on Evolution of Agent Development: Methodologies, Tools, Platforms and Languages, pages 101–107. Seneca Edizioni, 2008.

[70] Sotirios Liaskos, Alexei Lapouchnian, Yijun Yu, Eric Yu, and John Mylopoulos. On Goal-based Variability Acquisition and Analysis. In Proceedings of the 14th International

Requirements Engineering Conference, pages 76–85. IEEE Computer Society, 2006.

[71] Alexei Lapouchnian, Yijun Yu, Sotirios Liaskos, and John Mylopoulos. Requirements-Driven Design of Autonomic Application Software. In Proceedings of the 16th Conference of the Centre for Advanced Studies on Collaborative Research, pages 80–94. IBM, 2006.

[72] Betty Cheng, Rogerio de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vin- cenzo Grassi, Gabor Karsai, Holger Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In Betty Cheng, Rogerio de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, Software Engi- neering for Self-Adaptive Systems, volume 5525 of Lecture Notes in Computer Science, pages 1–26. Springer Berlin / Heidelberg, 2009.

[73] Daniel M. Berry, Betty H. C. Cheng, and Jia Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In Proceedings of the 11th International Workshop on Requirements Engineering Foundation for Software Quality, 2005.

[74] Heather J. Goldsby, Pete Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Danny Hughes. Goal-Based Modeling of Dynamically Adaptive System Requirements. In Proc. of the 15th Int. Workshop on Engineering of Computer-Based Systems, pages 36–45, 2008.

[75] Mohammed Salifu, Yijun Yu, and Bashar Nuseibeh. Specifying Monitoring and Switching Problems in Context. In Proceedings of the 15th International Requirements Engineering Conference, pages 211–220, 2007.

[76] Loris Penserini, Anna Perini, Angelo Susi, and John Mylopou- los. High variability design for software agents: Extending tropos. ACM Transactions on Autonomous and Adaptive Systems, 2(4):16, 2007.

[77] Raian Ali, Fabio Dalpiaz, and Paolo Giorgini. Modeling and Analizing Variability for Mobile Information System. In Proceedings of the 20th International conference on Advanced Information Sys- tems Engineering, pages 575–578. Lecture Notes in Computer Science, 2008.

[78] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A Goal Modeling Framework for Self-Contextualizable Software. In Proceedings of the 14th International Conference on Exploring Modeling Methods in Systems Analysis and Design, pages 326–338. Springer, 2009.

[79] Nauman A. Qureshi and Anna Perini. Engineering Adaptive Requirements. In Proceedings of the 4th International Workshop on Software Engineering for Adaptive and Self-Managing Systems, pages 126–131. ACM, 2009.

[80] Alexei Lapouchnian and John Mylopoulos. Modeling Domain Variability in Requirements Engineering with Contexts. In Proceedings of the 28th International Conference on Conceptual Modeling, pages 115–130. Springer-Verlag, 2009.

[81] Betty H. C. Cheng, Peter Sawyer, Nelly Bencomo, and Jon Whittle. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In Proc. of the 12th Int. Conf. on Model Driven Engineering Languages and Systems, pages 468–483, 2009.

[82] Mirko Morandini, Loris Penserini, and Anna Perini. Towards goal- oriented development of self-adaptive systems. In Proceedings of the 3rd International Workshop on Software Engineering for Adaptive and Self-Managing Systems, pages 9–16. ACM, 2008.

[83] Diana Lau and John Mylopoulos. Designing Web Services with Tropos. In Proceedings of the 2nd International Conference on Web Services, pages 306–313. IEEE Computer Society, 2004.

[84] Amy Lo and Eric Yu. From Business Models to Service-Oriented Design. In Proceedings of the 26th International Conference on Conceptual Modeling, pages 87–101. Springer-Verlag, 2007.

[85] Alexei Lapouchnian, Yijun Yu, and John Mylopoulos. Requirements-driven design and configuration management of business processes. In Proceedings of the 5th International Conference on Business Process Management, pages 246–261. Springer-Verlag, 2007.

[86] Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide), 2000 Edition, Chapter 5: Project Scope Management. Project Management Institute, Four Campus Boulevard, Newtown Square, PA 19073-3299, USA, 2000.

[87] K. Schmid. Scoping Software Product Lines – An Analysis of an Emerging Technology, Proceedings of the First Software Product Line Conference (SPLC'1), Kluwer, pp. 513-532, 2000.

[88] I. John, M. Eisenbarth, A Decade of Scoping – A Survey, In Proceedings of the 13th International Conference on Software Product Lines (SPLC), pp. 31 – 40, 2009.

[89] P. C. Clements. On the Importance of Product Line Scope. Software Product-Family Engineering, LNCS 2290, pp 102-111, 2002.

[90] A. Helferich, K. Schmid, G. Herzwurm. Product Management for Software Product Lines – An Overview, In: K. Kang, S. Park, V. Sugumaran (Eds.), Applied Software Product-Line Engineering, Auerbach Publications, 2009

[91] M. Khurum, T. Gorschek, K. Pettersson. Systematic Review of Solutions Proposed for Product Line Economics. Proceedings of the 12th International Software Product Line Conference (SPLC'08), pp.277-284, 2008.

[92] G. Böckle, P. Clements, J. D. McGregor, D. Muthig, K . Schmid. A Cost Model for Software Product Lines, Fifth International Workshop on Product Family Engineering (PFE-5), Siena, Italy, November 4-6, 2003.

[93] P. C. Clements, J. D. McGregor, S. G. Cohen. The Structured Intuitive Model for Product Line Economics (SIMPLE) (CMU/SEI-2005-TR-003), 2003.

[94] K. Schmid. A Comprehensive Product Line Scoping Approach and Its Validation. International Conference on Software Engineering (ICSE'24), IEEE Computer Society, pp. 593-603, 2002.

[95] K. Villela, J. Dörr, I. John. Evaluation of a Method for Proactively Managing the Evolving Scope of a Software Product Line. 16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2010), pp. 113-127, 2010.

[96] Patricia Collins Cornwell. HP domain analysis: Producing useful models for reusable software. Hewlett-Packard Journal, 47(4), August 1996.

[97] Jean-Marc DeBaud and Klaus Schmid. A practical comparison of major domain analysis approaches — towards a customizable domain analysis framework. In The 10th International Conference on Software Engineering & Knowledge Engineering, SEKE '98, pages 128–131, 1998.

[98] C. Salinesi, R. Mazo, D. Diaz, O. Djebbi. Using Integer Constraint Solving in Reuse Based Requirements Engineering. Proceedings of the 18th International Requirements Engineering Conference (RE'2010), pp. 243-251, 2010.

[99] Prasanna Padmanabhan, Robyn Lutz. Tool-supported verification of product line requirements, Automated Software Engineering Journal, Volume 12, Number 4, 447-465, Springer, 2005

[100] K. Lauenroth, K. Pohl. "Dynamic Consistency Checking of Domain Requirements in Product Line Engineering," Proceedings of the 16th International Requirements Engineering Conference (RE '08)., pp.193-202, 2008.

[101] Günter Halmans, Klaus Pohl, Communicating the Variability of a Software-Product Family to Customers. Software and System Modeling, Vol. 2, No. 1, pp. 15-36, 2003

[102] A. Monzon. A Practical Approach to Requirements Reuse in Product Families of On-Board Systems, Proceedings of the 16th International Requirements Engineering Conference (RE '08)., pp.223-228, 2008.

[103] O. Djebbi, C. Salinesi, D. Diaz. Deriving Product Line Requirements: the RED-PL Guidance Approach, Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC 2007), pp. 494-501, 2007.

[104] Sebastian Adam, Joerg Doerr, Michael Ehresmann, Pascal Wenzel; Incorporating SPL Knowledge into a Requirements Process for Information Systems – An Architecture-driven Tailoring Approach. First International Workshop on Product Line Requirements Engineering and Quality (PLREQ'10).

[105] A. Classen, A. Hubaux, P. Heymans. Analysis of Feature Configuration Workflows, Proceedings of the 17th International Requirements Engineering Conference (RE '09)., pp. 381-382, 2009.

[106] K. Schmid, H. Eichelberger, A Requirements-Based Taxonomy of Software Product Line Evolution, Electronic Communications of the EASST, Vol. 8, 2008.

[107] Klaus Schmid: Planning Software Reuse: A Disciplined Product Line Scoping Approach for Software Product Lines. PhD Thesis, University of Kaiserslautern, 2002.

[108] Böckle et al.: Software Produktlinien: Methoden, Einführung, Praxis, dpunkt-Verlag 2004; ISBN 3-89864-257-7.

[109] I. John, J. Knodel, T. Lehner, D. Muthig: A Practical Guide to Product Line Scoping, 10th International Software Product Line Conference (SPLC'06), IEEE 2006

[110] R. Kreuter, C. Lescher, A. Schreiber, C. Schwanninger, Applying a Cost Model for Product Lines: Experience Report, In Proceedings of the 12th International Software Product Lines Conference (SPLC'08), Second Volume, 2008.

[111] N.M. Josuttis*; SOA in Practice*; O'Reilly Publications, 2007.

[112] J. Davis; *Open Source SOA*; Mannings Publications 2009

[113] D. Chapell; *Enterprise Service Bus*; O 'Reilly, 2004

[114] Brian Jimerson, *Building a SOA Practice from the Ground Up*, http://www.oracle.com/technetwork/articles/soa/jimerson-soa-suite-case-study-239318.html

[115] Jörg Bartholdt, Bernd Franke, Christa Schwanninger, Michael Stal; *Combining Product Line Engineering and Service Oriented Architecture in Health Care Infrastructure Systems: Experience Report*, Proceedings of SPLC 2008; p.115-121; 2008

[116] Olaf Zimmermann, *An Architectural Decision Modeling Framework for Service Oriented Architecture Design*, Ph.D. Thesis, University of Stuttgart, 2009

[117] Cardoso, J. and Voigt, K. and Winkler, M.; Service engineering for the internet of services; Enterprise Information Systems, pp. 15-27, Springer, 2009

[118] A. G. J. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEE/IFP Conference on Software Architecture, WICSA, 2005.

[119] O. Zimmermann, T. Gschwind, J. Kuester, F. Leymann, and N. Schuster. Reusable architectural decision models for enterprise application development. In S. Overhage and C. Szyperski, editors, Quality of Software Architecture (QoSA) 2007, Lecture Notes in Computer Science, Boston, USA, July 2007. Springer-Verlag Berlin Heidelberg.

[120] O. Zimmermann, U. Zdun, T. Gschwind, and F. Leymann. Combining pattern languages and architectural decision models in a comprehensive and comprehensible design method. In Working IEEE/IFIP Conference on Software Architecture (WICSA) 2008, Vancouver, BC, Canada, February 2008.

[121] J. Tyree and A. Ackerman. Architecture decisions: Demystifying architecture. IEEE Software, 22(19–27), 2005.

[122] M. Ali Babar and P. Lago. Design decisions and design rationale in software architecture. Journal of Systems and Software 82(8): 1195-1197, 2009

[123] A. MacLean, R. Young, V. Bellotti, and T. Moran. Questions, options, and criteria: Elements of design space analysis. Human-Computer Interaction, 6(3–4):201–250, 1991.

[124] P. Kruchten, P. Lago, and H. Vliet. Building up and reasoning about architectural knowledge. In C. Hofmeister, editor, QoSA 2006 (Vol. LNCS 4214), pages 43–58, 2006.

[125] . Falessi, M. Becker, and G. Cantone. Design decicion rationale: Experiences and steps towards a more systematic approach. SIG-SOFT Software Eng. Notes 31 – Workshop on Sharing and Reusing Architectural Knowledge, 31(5), 2006.

[126] P. Kruchten, R. Capilla, and J. C. Duenas. The decision view's role in software architecture practice. IEEE Software, 26:36–42, 2009.

[127] P. Kruchten, "The 4+1 View Model of Architecture," IEEE Software, vol. 12, no. 6 , pp. 45–50, 1995.

[128] D. Schmidt and F. Buschmann. Patterns, frameworks, and middleware: Their synergistic relationships. In 25th International Conference on Software Engineering, pages 694–704, May 2003.

[129] F. Buschmann, K. Henney, D. C. Schmidt. Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages, Wiley, 2007.

[130] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[131] N. Harrison, P. Avgeriou, and U. Zdun. Using Patterns to Capture Architectural Decisions. IEEE Software, pages 38-45, IEEE, July/Aug., 2007.

[132] L. Bass, M. Klein, F. Bachmann. Quality Attribute Design Primitives and the Attribute Driven Design Method, Proceedings of the 4th International Workshop on Product Family Engineering (PFE-4), 2002.

[133] J. Bosch. Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach, Addison-Wesley, 2000.

[134] M. Matinlassi. Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA. Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), pp. 127 – 136, 2004.

[135] P. America, H. Obbink, J. Muller, R. van Ommering. COPA: A Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products, Proceedings of the First Conference on Software Product Line Engineering (SPLC 2000), 2000.

[136] D. Weiss, C. Lai, R. Tau. Software product-line engineering: a family-based software development process, Addison-Wesley, 1999.

[137] C. Atkinson, J. Bayer, C. Bunser, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Peach, J. Wust, J. Zettel. Component-based product line engineering with UML, Addison-Wesley, 2002.

[138] M. Matinlassi, E. Niemelä, L. Dobrica. Quality-driven architecture design and quality analysis method, A revolutionary initiation approach to a product line architecture, VTT Technical Research Centre of Finland, Espoo, 2002.

[139] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid,T. Widen, J.-M. DeBaud. PuLSE: A Methodology to Develop Software Product Lines, Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99), pp. 122–131, 1999.

[140] P. Knauber, D. Muthig, K. Schmid, T.Widen. Applying Product Line Concepts in Small and Medium-Sized Companies, IEEE SOFTWARE, Vol. 17, pp. 88 – 95, 2000.

[141] P. Heymans, J. C. Trigaux. Software Product Line: State of the art. Technical report for PLENTY project, Institut d'Informatique FUNDP, Namur, 2003.

[142] J. Mendling and C. Simon. Business Process Design by View Integration. In Business Process Management Workshops, volume 4103 of LNCS, pages 55–64. Springer, 2006.

[143] R. Davis. Business Process Modelling with ARIS: a Practical Guide. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[144] E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In Business Process Management, pages 82–97, 2004.

[145] B. Axenath, E. Kindler, and V. Rubin. An Open and Formalism Independent Meta-Model for Business Processes. In Proc. of the Workshop on Business Process Reference Models, pages 45–59, 2005.

[146] B. Axenath, E. Kindler, and V. Rubin. AMFIBIA: A Meta-Model for the Integration of Business Process Modelling Aspects. International Journal of Business Process Integration and Management, 2(2):120–131, 2007.

[147] ISO/IEC 10746-3 Open Distributed Processing – Reference Model: Architecture. http://standards.iso.org/ittf/PubliclyAvailableStandards/s020697_ISO_IEC_10746-3_1996(E).zip, Sept. 1996.

[148] R. J. Mayer, C. P. Menzel, M. K. Painter, P. S. de Witte, T. Blinn, and B. Perakath. Integrated DEF-inition for Process Description Capture Method Report. http://www.idef.com/pdf/Idef3_fn.pdf, Sept. 1995.

[149] H. Tran, T. Holmes, U. Zdun, and S. Dustdar. Modeling Process-Driven SOAs - a View-Based Approach. In J. Cardoso and W. van der Aalst, editors. Handbook of Research on Business Process Modeling, IGI Global, Hershey, USA, 2009.

[150] H. Tran, U. Zdun, and S. Dustdar. VbTrace: Using View-based and Model-driven Development to Support Traceability in Process-driven SOAs. Software & System Modeling, Springer, 2009.

[151] R. Hamadi and B. Benatallah, "A petri net-based model for web ser- vice composition," in ADC '03: Proceedings of the 14th Australasian database conference. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2003, pp. 191–200.

[152] M. Dumas and A. H. M. ter Hofstede, "Uml activity diagrams as a workflow specification language," in UML01: Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. London, UK: Springer-Verlag, 2001, pp. 76–90.

[153] T. Mitra, "Business-driven development," http://www-128.ibm.com/developerworks/webservices/library/ws-bdd/index.html, 2005.

[154] D. Skogan, R. Gronmo, and I. Solheim, "Web service composition in uml," in EDOC '04: Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International. Washington, DC, USA: IEEE Computer Society, 2004, pp. 47–57.

[155] R. Gronmo and M. C. Jaeger, "Model-driven semantic web service com- position," in APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference. Washington, DC, USA: IEEE Computer So- ciety, 2005, pp. 79–86.

[156] J. Koehler, R. Hauser, R. Hauser, S. Kapoor, F. Y. Wu, and S. Ku- maran, "A model-driven transformation method," in EDOC '03: Pro- ceedings of the 7th International Conference on Enterprise Distributed Object Computing. Washington, DC, USA: IEEE Computer Society, 2003, p. 186.

[157] R. Hauser and J. Koehler, "Compiling process graphs into executable code," in GPCE, 2004, pp. 317–336.

[158] C. Ouyang, W. M. van der Aalst, M. Dumas, and A. H. ter Hofstede, "Translating bpmn to bpel," BPM Center Report, Tech. Rep. BPM- 06-02, 2006.

[159] B. Orriens, J. Yang, and M. P. Papazoglou, "Model driven service com- position," in ICSOC: Proceedings of the International Conference on Service-Oriented Computing, 2003.

[160] D. Muthig, T. Patzke, Generic Implementation of Product Line Components, In Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World, pp. 313 – 329, 2002.

[161] J. van Gurp, J. Savolainen, Service Grid Variability Realization, Proceedings of 10th International Software Product Line Conference (SPLC 2006), 85 – 94, 2006.

[162] M. Völter, I. Groher. Product Line Implementation using Aspect-Oriented and Model-Driven Software Development, Proceedings of 11th International Software Product Line Conference (SPLC 2007), 233-242, 2007.

[163] A. van der Hoek. Design-time product line architectures for any-time variability. Science of Computer Programming, 53(30):285-304, 2004.

[164] K. Schmid, H. Eichelberger. Model-Based Implementation of Meta-Variability Constructs: A Case Study using Aspects, Second International Workshop on Variability Modeling of Software-Intensive Systems, ICB-Research Report No. 22, ISSN 1860-2770, pp. 63-71, 2008.

[165] S. Hallsteinsen, M. Hinchey, S. Park, K. Schmid. Dynamic Software Product Lines. IEEE Computer, Vol. 41, No. 4, pp. 93 – 95, 2008.

[166] V. Buregio, E. Almeida, S. Meira. Characterizing Dynamic Software Product Lines – A Preliminary Mapping Study, Proceedings of the 4th International Workshop on Dynamic Software Product Lines (DSPL 2010) at (SPLC 2010), pp. 53 – 60, 2010.

[167] J. Lee and K. Kang. A feature-oriented approach to developing dynamically reconfigurable products in product line engineering. In SPLC '06: Proceedings of the 10th International on Software Product Line Conference, pages 131–140, 2006.

[168] I. Montero, J. Peña, and A. Ruiz-Cortés. Towards visualisation and analysis of runtime variability in execution time of business information systems based on product line. In *Proceedings of the Second International Workshop on Variability Modelling of Software-intensive Systems (VAMOS'08)*, 2008. ICB-Research Report No. 22.

[169] W Kongdenfha, R Saint-Paul, B Benatallah. An aspect-oriented framework for service adaptation. In Proceedings of the International Conference on Service-Oriented Computing (ICSOC), 2006

[170] D. Ardagna, M. Comuzzi, E. Mussi. Paws: A framework for executing adaptive web-service processes. In IEEE Internet Computing vol. 24 no. 6, 2007.

[171] M. Di Penta, R. Esposito, M. Villani. WS Binder: a framework to enable dynamic binding of composite web services. In SOSE'06: Proceedings of the 2006 International Workshop on Service-Oriented Software Engineering, 2006.

[172] A. Erradi, P. Maheshwari, V. Tosic. Policy-driven middleware for self-adaptation of web services compositions. In Middleware'06: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware, 2006.

[173] O. Moser, F. Rosenberg, S. Dustdar. Non-intrusive monitoring and service adaptation for WS-BPEL. In WWW'08: Proceedings of the World Wide Web Conference, 2008

[174] A. Michlmayr, F. Rosenberg, P. Leitner, S. Dustdar. End-to-End Support for QoS-Aware Service Selection, Invocation and Mediation in VRESCo. In IEEE Transactions on Services Computing (TSC), 2010.

[175 F. Irmert, T. Fischer, K. Meyer-Wegener. Runtime Adaptation in a Service-Oriented Component Model. SEAMS '08, In proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems, 2008.

[176] B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features – Enhancing flexibility in process-aware information systems. In Data & Knowledge Engineering, 2008

[177] D. Karastoyanova, F. Leymann, and J. Nitzsche. Parameterized BPEL Processes: Concepts and Implementation. In Proceedings of the 4th International Conference on Business Process Management (BPM), 2006

[178] A. Charfi and M. Mezini. Aspect-Oriented Web Service Composition with AO4BPEL. In Proceedings of the European Conference on Web Services (ECOWS), 2004

[179] N. C. Narendra, K. Ponnalagu, J. Krishnamurthy, R. Ramkumar. Run-Time Adaptation of Non-functional Properties of Composite Web Services Using Aspect-Oriented Programming, ICSOC '07, in proceedings of the 5th international conference on Service-Oriented Computing, 2007.

[180] D. Karastoyanova, and F. Leymann. BPEL'n'Aspects: Adapting Service Orchestration Logic. In Proceedings of the 2009 International Conference on Web Services (ICWS), 2009.

[181] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. Monitoring, Prediction and Prevention of SLA Violations in Composite Services, In Proceedings of the International Conference on Web Services (ICWS), 2009.

[182] P. Leitner, B. Wetzstein, D. Karastoyanova, W. Hummer, S. Dustdar, and F. Leymann. Preventing SLA Violations in Service Compositions Using Aspect-Based Fragment Substitution. In Proccedings of the International Conference on Service-Oriented Computing (ICSOC), 2009.

[183] P. Rossi, Z. Tari. Software Adaptation for Service-Oriented Systems. MW4SOC '06, in proceedings of the 1st workshop on Middleware for Service Oriented Computing, 2006.

[184] S. H. Chang, H. J. La, S. D. Kim. A Comprehensive Approach to Service Adaptation. SOCA '07, In proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, 2007.

[185] J. Harney, P. Doshi. Speeding up Adaptation of Web Service Compositions Using Expiration Times. WWW '07, In proceedings of the 16th international conference on World Wide Web, 2007.

[186] G. Ortiz, A. García de Prado. Web Service Adaptation. ICIW '10, In proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services, 2010.

[187] M. Sidibé, A. Mehaoua. Service Monitoring System for Dynamic Service Adaptation in Multi-domain and Heterogeneous Networks. WIAMIS '08, In proceedings of the 2008 Ninth International Workshop on Image Analysis for Multimedia Interactive Services, 2008.

[188] K. Wang, M. Dumas, C. Ouyang, J. Vayssière. The Service Adaptation Machine .ECOWS '08, In proceedings of the 2008 Sixth European Conference on Web Services, 2008

[189] B. Wu, S. Deng, J. Wu, Y. Li, L. Kuang, J. Yin. Service Behavioral Adaptation based on Dependency Graph. APSCC '08, In proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference, 2008

[190] L. González, R. Ruggia. Towards Dynamic Adaptation within an ESB-based Service Infrastructure Layer. MONA '10, In proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond, 2010.

[191] A. Bucchiarone, R. Kazhamiakin, C. Cappiello, E. di Nitto, V. Mazza. A context-driven adaptation process for service-based applications. PESOS '10, In proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems, 2010.

[192] H. Gomaa, K. Hashimoto, M. Kim, S. Malek, D. A. Menascé. Software Adaptation Patterns for Service-Oriented Architectures. SAC '10, In proceedings of the 2010 ACM Symposium on Applied Computing, 2010.

[193] E.A. Marks: Service Oriented Architecture Governance for the Service Driven Enterprise. Wiley, Hoboken NJ, 2008.

[194] Soumya Simanta et al.: A Scenario-Based Technique for Developing SOA Technical Governance. TECHNICAL NOTE CMU/SEI-2009-TN-009, Carnegie Mellon University, Pittsburgh, 2009.

[195] Jan Berhardt, Detlef Seese: A Conceptual Framework for the Governance of Service-Oriented Architectures, in: Service-Oriented Computing – ICSOC 2008 Workshops, Lecture Notes in Computer Science, 2009, Volume 5472/2009, 327-338

[196] Reference Model for Service Oriented Architecture 1.0, OASIS 2005-2006, download from http://docs.oasis-open.org/soa-rm/v1.0/

[197] Fazilat Hojaji, Mohammad Reza Ayatollahzadeh Shirazi: AUT SOA Governance: A New SOA Governance Framework Based on COBIT, 2010 3rd International Conference on Computer Science and Information Technology(2010) vol. 8 p. 403 - 408

[198] de Leusse, P.; Dimitrakos, T.; Brossard, D.: A Governance Model for SOA, 2009 IEEE International Conference on Web Services (2009) vol. p.1020 – 1027

[199] The Open Group: SOA Governance Framework, Draft Technical Standard, 2009, https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?catalogno=c093

[200] Niemann, Miede, Johannsen, Repp, Steinmetz: Structuring SOA Governance. International Journal on IT/Business Alignment and Governance, 1(1), 58-75, January-March 2010

[201] A. Birukou, V. D'Andrea, F. Leymann, J. Serafinski, P. Silveira, S. Strauch, M. Tluczek: An Integrated Solution for Runtime Compliance Governance in SOA. Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC10), USA, 2010.

[202] Service Lifecycle Governance with IBM WebSphere Service Registry and Repository, Advanced Lifecycle Edition, Armonk NY, September 2009.

[203] Carzaniga, A., Fuggetta, A., Hall, R.S., van der Hoek, A., Heimbigner, D., Wolf, A. L.: A Characterization Framework for Software Deployment Technologies, TR CU-CS-857-98, University of Colorado, April 1998

[204] Hofman, A. and Neubauer, B. (2005) "Deployment and Configuration of Distributed Systems", System Analysis and Modeling, Lecture Notes in Computer Science, Vol. 3319, pp. 1-16.

[205] Vanish Talwar, Dejan Milojicic, Qinyi Wu, Calton Pu, Wenchang Yan, and Gueyoung Jung. 2005. Approaches for Service Deployment. *IEEE Internet Computing* 9, 2 (March 2005), 70-80. DOI=10.1109/MIC.2005.32 http://dx.doi.org/10.1109/MIC.2005.32

[206] Vanish Talwar, Qinyi Wu, Calton Pu, Wenchang Yan, Gueyoung Jung, and Dejan Milojicic. 2005. Comparison of Approaches to Service Deployment. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems* (ICDCS '05). IEEE Computer Society, Washington, DC, USA, 543-552.

[207] [OMG 2004] Object Management Group: Common Object Request Broker Architecture: Core Specification, 2004

[208] Fahringer T., Jugravu A. JavaSymphony: A New Programming Paradigm to Control and to Synchronize Locality, Parallelism, and Load Balancing for Parallel and Distributed Computing. Concurrency and Computation: Practice and Experience 2002; 17,7-8:1005-1025

[209] Tilevich E., Smaragdakis Y. J-Orchestra: Automatic Java Application Partitioning. In: Proc. European Conference on Object-Oriented Programming (ECOOP), Malaga, 2002.

[210] Kirby, G. N. C., Walker, S. M., Norcross, S. J., Dearle, A., & Eisenbach, S. (2005). A Methodology for Developing and Deploying Distributed Applications. *3rd International Working Conference on Component Deployment CD2005* (Vol. 3798, pp. 37-51). Springer.

[211] Object Management Group: Model Driven Architecture (MDA), OMG document ormsc/01-07-01, July 2001

[212] Object Management Group: Deployment & Configuration of Component-based Distributed Applications Specification, OMG document ptc/04-05-14, May 2004

[213] Hnetynka, P., "A model-driven environment for component deployment," *Software Engineering Research, Management and Applications, 2005. Third ACIS International Conference on* , vol., no., pp. 6- 13, 11-13 Aug. 2005 doi: 10.1109/SERA.2005.12

[214] Ang Tan Fong, Ling Teck Chaw, Phang Keat Keong, and Por Lip Yee. 2009. Automatic Web Services Deployment. In *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering - Volume 07* (CSIE '09), Vol. 7. IEEE Computer Society, Washington, DC, USA, 315-319.

[215] Sam Malek, Marija Mikic-Rakic, and Nenad Medvidovic. A Decentralized Redeployment Algorithm for Improving the Availability of Distributed Systems. In Proc. of *the 3rd Int. Working Conference on Component Deployment (CD 2005)*, Grenoble, France, Nov. 2005.

[216] Gan Deng, Douglas C. Schmidt, and Aniruddha Gokhale. 2008. CaDAnCE: A Criticality-Aware Deployment and Configuration Engine. In *Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing* (ISORC '08). IEEE Computer Society, Washington, DC, USA, 317-321. DOI=10.1109/ISORC.2008.58 http://dx.doi.org/10.1109/ISORC.2008.58

[217] Object Management Group: Deployment and Configuration of Component-based Distributed Applications Specification. OMG document formal/06-04-02, April 2006

[218] Inderjeet Singh, Beth Stearns, Mark Johnson, and the Enterprise Team. Designing Enterprise Applications with the J2EETM Platform, Second Edition, J2EE BluePrints 2002.

[219] Carles Pairot Gavalda, Pedro Garcia Lopez, and Ruben Mondejar Andreu. 2007. Deploying Wide-Area Applications Is a Snap. *IEEE Internet Computing* 11, 2 (March 2007), 72-79. DOI=10.1109/MIC.2007.31 http://dx.doi.org/10.1109/MIC.2007.31

[220] A. Heydarnoori, F. Mavaddat, F. Arbab. Towards an Automated Deployment Planner for Composition of Web Services as Software Components. Electronic Notes in Theoretical Computer Science (ENTCS), 160, pp. 239-253. 2006.

[221] Arbab, F. “Reo: A Channel-based Coordination Model for Component Composition,” Mathematical Structures in Computer Science, 14, 3 (June 2004), 329-366.

[222] J. de Meer, A. Rennoch, J. Burmeister. CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering - Volume 1, 1993

[223] R. Pignaton, V. Villagra, J. I. Asensio, J. J. Berrocal. Developing QoS-aware Component-Based Applications Using MDA Principles. In proceedings: EDOC '04 Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International, 2004

[224] C. Wang, G. Wang, A. Chen, H. Wang, Y. Pierce, C. Fung, S. Uczekaj. A Policy-Based Approach for QoS Specification and Enforcement in Distributed Service-Oriented Architecture. SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing - Volume 01, 2005

[225] Sabata, B., Chatterjee, S.; Davis, M.; Sydir, J.J.; Lawrence, T.F. Taxonomy for QoS specifications. Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on, 1997

[226] A. Campbell, C. Aurrecoechea and L. Hauw. A Review of QoS Architectures. Proceedings of the 4th IFIP International workshop on Quality of Service (IWQS '96), Paris, March, 1996

[227] A. Marefat. Dynamic Management of Multimedia Applications and Multilevel QOS Specification. In proceedings: IEEE International Conference on Multimedia Computing and Systems '97. 1997

[228] S. Frølund, J. Koistinen. Quality of services specification in distributed object systems design. COOTS'98: Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems - Volume 4, 1998

[229] Y. Matsui, S. Kihara, A. Mitsuzawa, S. Moriai, H. Tokuda. An Extensible Object Model for QoS Specification in Adaptive QoS Systems. ISORC '99: Proceedings of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 1999

[230] I. Widya, R. E. Stap, L. J. Teunissen, B. F. Hopman. On the End-User QoS-Awareness of a Distributed Service Environment. PROMS 2001: Proceedings of the 6th International Conference on Protocols for Multimedia Systems, 2001

[231] ISO/IEC JTC1/SC21 N13236, "Information Technology – Quality of Service – Framework", Geneva, 1997

[232] Jan Øyvind Aagedal and Earl F. Ecklund. Modelling QoS: Towards a UML Profile. Lecture Notes in Computer Science, Volume 2460/2002, 65-75, DOI: 10.1007/3-540-45800-X_22, 2002

[233] C. Koliver, K. Nahrstedt, J.-M. Farines, J. da Silva Fraga and S. Aparecida Sandri. Specification, Mapping and Control for QoS Adaptation. Real-Time Systems archive Volume 23 Issue 1/2, July-September 2002

[234] B. Bordbar, J. Derrick, G. Waters. Using UML to specify QoS constraints in ODP. Journal: Computer Networks: The International Journal of Computer and Telecommunications Networking, 2002

[235] Object Management Group. UML Profile for Enterprise Distributed Object Computing (EDOC) Specification OMG Document: ptc/02-02-05. (2002).

[236] J. Jin, K. Nahrstedt. QoS specification languages for distributed multimedia applications: a survey and taxonomy. Journal IEEE MultiMedia Volume 11 Issue 3, July 2004

[237] M.A. de Miguel, M.T. Higuera. Runtime management of quality specification for QoS-aware components. In proceedings 30th Euromicro Conference, 2004

[238] G. Dobson, R. Lock, I. Sommerville. QoSOnt: a QoS ontology for service-centric systems. In proceeding: EUROMICRO '05 Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, 2005

[239] A. Lopes, F. Amaro, G. Elias, G. Lemos, M. F. Magalhaes. QoS Specification and Management in a Middleware for Distributed Multimedia Systems. AINA '06: Proceedings

of the 20th International Conference on Advanced Information Networking and Applications - Volume 1, 2006.

[240] A. C. Prudêncio, M. L. Sheibel, R. Willrich, S. Tazi, G. Sancho. Application and Network QoS Mapping Using an Ontology-based Approach. In proceedings: CTRQ'10 Proceedings of the 2010 Third International Conference on Communication Theory, Reliability, and Quality of Service, 2010.

[241] Diana Berberova and Boyan Bontchev. 2009. Design of service level agreements for software services. In *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing* (CompSysTech '09), Boris Rachev and Angel Smrikarov (Eds.). ACM, New York, NY, USA, , Article 26 , 6 pages.

[242] James Skene, D. Davide Lamanna, and Wolfgang Emmerich. 2004. Precise Service Level Agreements. In *Proceedings of the 26th International Conference on Software Engineering* (ICSE '04). IEEE Computer Society, Washington, DC, USA, 179-188.

[243] Alessio Gambi, Giovanni Toffetti, and Mauro Pezzè. 2010. Protecting SLAs with surrogate models. In *Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems* (PESOS '10). ACM, New York, NY, USA, 71-77.

[244] Philipp Leitner, Anton Michlmayr, Florian Rosenberg, and Schahram Dustdar. 2010. Monitoring, Prediction and Prevention of SLA Violations in Composite Services. In *Proceedings of the 2010 IEEE International Conference on Web Services* (ICWS '10). IEEE Computer Society, Washington, DC, USA, 369-376.

[245] Vinod Muthusamy, Hans-Arno Jacobsen, Tony Chau, Allen Chan, and Phil Coulthard. 2009. SLA-driven business process management in SOA. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research* (CASCON '09), Patrick Martin, Anatol W. Kark, and Darlene Stewart (Eds.). ACM, New York, NY, USA, 86-100.

[246] Paschke, A., Schnappinger-Gerull, E.. *A Categorization Scheme for SLA Metrics. Multi-Conference Information Systems (MKWI)*. 2006. Passau, Germany.

[247] Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R., A Service Level Agreement Language for Dynamic Electronic Services, *Journal of Electronic Commerce Research*, Volume 3, Issue 1&2, Kluwer Academic Publishers, March, 2003

[248] Keller, A., Ludwig, H., The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services, *Journal of Network and Systems Management, Special Issue on E-Business Management*, Volume 11, Number 1, Plenum Publishing Corporation, March, 2003

[249] H. Ludwig, A. Keller, A. Dan, R. Franck, and R.P. King, Web Service Level Agreement (WSLA) *Language Specification*, IBM Corporation, July 2002

[250] Vladimir Tosic, Kruti Patel, and Bernard Pagurek. 2002. WSOL - Web Service Offerings Language. In *Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web* (CAiSE '02/ WES '02). Springer-Verlag, London, UK, UK, 57-67.

[251] Tosic, V., Pagurek, B., Esfandiari, B., Patel, K., Ma, W.: Web Service Offerings Language (WSOL) and Web Service Composition Management (WSCM). In Proc. of the OOWS'02 (Object-Oriented Web Services) workshop at OOPSLA 2002 (Seattle, USA, Nov. 2002)

[252] Adrian Paschke. 2005. RBSLA A declarative Rule-based Service Level Agreement Language based on RuleML. In *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06) - Volume 02* (CIMCA '05), Vol. 2. IEEE Computer Society, Washington, DC, USA, 308-314.

[253] Web Services Agreement Specification (WS-Agreement). Specification from the Open Grid Forum (OGF). http://www.ogf.org/documents/GFD.107.pdf

[254] W3C. Web Service Modeling Ontology (WSMO). W3C Member Submission 3 June 2005.

[255] Ioan Toma, Douglas Foxvog, and Michael C. Jaeger. 2006. Modeling QoS characteristics in WSMO. In *Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)* (MW4SOC '06). ACM, New York, NY, USA, 42-47.

[256] [Lamanna et al. 2003] D. Davide Lamanna, James Skene, and Wolfgang Emmerich. 2003. SLAng: A Language for Defining Service Level Agreements. In *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems* (FTDCS '03). IEEE Computer Society, Washington, DC, USA, 100-110.

[257] Franco Raimondi, James Skene, and Wolfgang Emmerich. 2008. Efficient online monitoring of web-service SLAs. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (SIGSOFT '08/FSE-16). ACM, New York, NY, USA, 170-180.

[258] James Skene, Allan Skene, Jason Crampton, and Wolfgang Emmerich. 2007. The monitorability of service-level agreements for application-service provision. In *Proceedings of the 6th international workshop on Software and performance* (WOSP '07). ACM, New York, NY, USA, 3-14.

[259] N. J. Dingle and W. J. Knottenbelt and L. Wang. Service Level Agreement Specification Compliance Prediction and Monitoring with Performance Trees, ESM08.

[260] Torvald Riegel. A generalized approach to runtime monitoring for real-time systems. Master's thesis, Technische Universität Dresden, 2005.

[261] K. Bratanis, D. Dranidis, and A.J.H. Simons. An extensible architecture for run-time monitoring of conversational web services. In *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond*, pages 9–16. ACM, 2010.

[262] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. Comprehensive QoS monitoring of Web services and event-based SLA violation detection. In *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing*, pages 1–6. ACM, 2009.

[263] SS Yau, N. Ye, H. Sarjoughian, and D. Huang. Developing Service-Based Software Systems with QoS Monitoring and Adaptation. In *Future Trends of Distributed Computing Systems, 2008. FTDCS'08. 12th IEEE International Workshop on*, pages 74–80. IEEE, 2008.

[264] Microsoft MSDN. Wcf performance counters. http://msdn.microsoft.com/en-us/library/ms735098.aspx.

[265] H. Zadeh and M.A. Seyyedi. Qos monitoring for web services by Time Series Forecasting. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 5, pages 659–663. IEEE, 2010.

[266] L. Fei, Y. Fangchun, S. Kai, and S. Sen. A policy-driven distributed framework for monitoring quality of web services. In *Web Services, 2008. ICWS'08. IEEE International Conference on*, pages 708–715. IEEE, 2008.

[267] A. Lazovik, M. Aiello, and M. Papazoglou. Associating assertions with business processes and monitoring their execution. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 94–104. ACM, 2004.

[268] L. Baresi, C. Ghezzi, and S. Guinea. Smart monitors for composed services. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 193–202. ACM, 2004.

[269] L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. *Service-Oriented Computing-ICSOC 2005*, pages 269–282, 2005.

[270] M. Lohmann, L. Mariani, and R. Heckel. A model-driven approach to discovery, testing and monitoring of web services. *Test and Analysis of Web Services*, pages 173–204, 2007.

[271] Z. Li, Y. Jin, and J. Han. A runtime monitoring and validation framework for web service interactions. In *Software Engineering Conference, 2006. Australian*, pages 10–79. IEEE, 2006.

[272] J. Simmonds, Yuan Gan, M. Chechik, S. Nejati, B. O'Farrell, E. Litani, and J. Waterhouse. Runtime monitoring of web service conversations. *IEEE Transactions on Services Computing*, 2(3):223–244, 2009.

[273] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Malhotra, et al. Web services policy framework (ws-policy). *Version*, 1(2):2003–2006, 2006.

[274] B.N. Grosof. Representing e-commerce rules via situated courteous logic programs in ruleml* 1. *Electronic Commerce Research and Applications*, 3(1):2–20, 2004.

[275] Khaled Mahbub and George Spanoudakis. A framework for requirents monitoring of service based systems. In *Proceedings of the 2nd international conference on Service oriented computing*, ICSOC '04, pages 84–93, New York, NY, USA, 2004. ACM.

[276] K. Mahbub and G. Spanoudakis. Run-time monitoring of requirements for systems composed of web-services: initial implementation and evaluation experience. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pages 257 – 265 vol.1, 2005.

[277] W.M.P. Aalst and M. Pesic. Specifying and monitoring service flows: Making web services process-aware. *Test and Analysis of Web Services*, pages 11–55, 2007.

[278] F.H. Zulkernine, P. Martin, and K. Wilson. A Middleware Solution to Monitoring Composite Web Services-based Processes. In *2008 IEEE Congress on Services Part II*, pages 149–156. IEEE, 2008.

[279] M. Alodib and B. Bordbar. A model-based approach to Fault diagnosis in Service oriented Architectures. In *Web Services, 2009. ECOWS'09. Seventh IEEE European Conference on*, pages 129–138. IEEE, 2009.

[280] L. Baresi, S. Guinea, O. Nano, and G. Spanoudakis. Comprehensive Monitoring of BPEL Processes. *Internet Computing, IEEE*, 14(3):50–57, 2010.

[281] B. Wetzstein, D. Karastoyanova, O. Kopp, F. Leymann, and D. Zwink. Cross-organizational process monitoring based on service choreographies. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 2485–2490. ACM, 2010.

[282] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. Architecture-Based Runtime Software Evolution. In Proceedings of the 20th International Conference on Software Engineering, pages 177–186. IEEE Computer Society, 1998.

[283] David Garlan, Shang-Weng Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. IEEE Computer, 37(10):46–54, 2004.

[284] Franck Fleurey, Vegard Dehlen, Nelly Bencomo, Brice Morin, and Jean-Marc Jézéquel. Modeling and Validating Dynamic Adaptation. In Proceedings of the 5th International Workshop Models@runtime, pages 97–108. Springer-Verlag, 2008.

[285] Daniel Sykes, William Heaven, Jeff Magee, and Jeff Kramer. Exploiting non-functional preferences in architectural adaptation for self-managed systems. In Proceedings of the 25th Symposium on Applied Computing, pages 431–438. ACM, 2010.

[286] Hossein Tajalli, Joshua Garcia, George Edwards, and Nenad Medvidovic. PLASMA: A Plan-based Layered Architecture for Soft- ware Model-driven Adaptation. In Proceedings of the 25th Inter- national Conference on Automated Software Engineering, page to appear. ACM, 2010.

[287] Jeff Kramer and Jeff Magee. Self-Managed Systems: an Architectural Challenge. In Proc. of Future of Software Engineering, pages 259–268, 2007.

[288] Luciano Baresi and Sam Guinea. Self-supervising BPEL Processes. IEEE Trans. on Software Engineering, 2011.

[289] Luciano Baresi, Domenico Bianculli, Carlo Ghezzi, Sam Guinea, and Paola. Spoletini. Validation of Web Service Compositions. IET Software, 1(6):219–232, 2007.

[290] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In Proceedings of the 11th Euro- pean Conference on Object-Oriented Programming, pages 220–242. Springer, 1997.

[291] Alexander Keller and Heiko Ludwig. Defining and Monitoring Service-Level Agreements for Dynamic e-Business. In Proceedings of the 16th USENIX conference on System administration, pages 189–204. USENIX Association, 2002.

[292] Akhil Sahai, Vijay Machiraju, Mehmet Sayal, Aad P. A. van Moorsel, and Fabio Casati. Automated SLA Monitoring for Web Services. In Proceedings of the 13th International Workshop on Distributed Systems: Operations and Management, pages 28–41. Springer-Verlag, 2002.

[293] Abdelkarim Erradi, Piyush Maheshwari, and Vladimir Tosic. WS-Policy based Monitoring of Composite Web Services. In Proceed- ings of the 5th European Conference on Web Services, pages 99– 108. IEEE Computer Society, 2007.

[294] W3C. Web Services Policy 1.2. http://www.w3.org/Submission/ WS-Policy/.

[295] Vladimir Tosic, Abdelkarim Erradi, and Piyush Maheshwari. WS- Policy4MASC - A WS-Policy Extension Used in the MASC Middleware. In Proceedings of the 5th European Conference on Web Services, pages 458–465. IEEE Computer Society, 2007.

[296] Marco Comuzzi and George Spanoudakis. A Framework for Hierarchical and Recursive Monitoring of Service Based Systems. In Proceedings of the 4th International Confer-

ence on Internet and Web Applications and Services, pages 383–388. IEEE Computer Society, 2009.

[297] Marco Comuzzi and George Spanoudakis. Dynamic set-up of Monitoring Infrastructures for Service Based Systems. In Proceedings of the 25th Symposium on Applied Computing, pages 2414–2421. ACM, 2010.

[298] Massimiliano Colombo, Elisabetta Di Nitto, and Marco Mauri. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In Proceedings of the 4th International Conference on Service Oriented Computing, pages 191–202, 2006.

[299] H. Kett, K. Voigt, G. Scheithauer, J. Cardoso. Service engineering in business ecosystems; Proceedings of the XVIII. International RESER Conference, pp. 25-26; 2008

[300] H. Kett, G. Scheithauer, N. WEINER, A. WEISBECKER. Integrated Service Engineering (ISE) for Service Ecosystems: An Interdisciplinary Methodology for the Internet of Services; Proceedings of the eChallenges e-2009 Conference ; 2009

[301] J. A. Zachman. A Framework for Information Systems Architecture; IBM Systems Journal, pp. 276-292; 1987

[302] M. Voelter, E. Visser. Product Line Engineering using Domain-Specific Languages; In Proceedings of the 13th International Software Product Line Conference; 2009

[303] S. D. Kim, H. G. Min, J. S. Her, S. H. Chang. DREAM: A Practical Product Line Engineering Using Model Driven Architecture. In Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05) Volume 2 - Volume 02; 2005

[304] A. Rummler, B. Grammel, C. Pohl. Improving traceability in model-driven development of business applications; Traceability Workshop (ECMDA-TW); 2007

[305] MDPLE. 3rd International Workshop on Model-Driven Product Line Engineering, Brimingham, 6th June 2011. http://sites.lero.ie/mdple2011/.

[306] K. Czarnecki, S. Helsen, and U.Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. In Software Process: Improvement and Practice, volume 10, opages 143-169, 2005.

[307] M. Rosenmüller, N. Siegmund, T. Thüm, and G. Saake. Multi-dimensional variability modeling. In 5th International Workshop on Variability Modelling of Software-intensive Systems, pages 11–20, 2011.

[308] A. Haber, H. Rendel, B. Rumpe, I. Schaefer, F. van der Linden. Hierarchical Variability Modeling for Software Architectures. In Proceedings Software Product Line Conference (SPLC 2011), 2011.

[309] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The Koala Component Model for Consumer Electronics Software. IEEE Computer, pp 78 – 85, 2000.

[310] R. van Ommering. Software Reuse in Product Populations. IEEE Transactions on Software Engineering, vol. 31, pp 537 – 550, 2005.

[311] A. Haber, H. Rendel, B. Rumpe, and I. Schaefer. Delta Modeling for Software Architectures. Modellbasierte Entwicklung Eingebetteter Systeme (MBEES), 2011.

[312] J. Pérez, J. Díaz, C. Costa-Soria, and J. Garbajosa. Plastic Partial Components: A Solution to Support Variability in Architectural Components. Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture 2009 (WICSA 2009), pp. 221 – 230, 2009.

[313] M. Matinlassi, E. Niemelä, L. Dobrica. Quality-driven architecture design and quality analysis method, A revolutionary initiation approach to a product line architecture, VTT Technical Research Centre of Finland, Espoo, 2002.

[314] www.biglever.com, visited October 2009.

[315] www.pure-systems.com, visited October 2009.

[316] K. Schmid, R. Rabiser, P. Grünbacher. A Comparison of Decision Modeling Approaches in Product Lines. 5th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2011), pp. 119 – 126, 2011

[317] D. Dhungana, T. Neumayer, P. Grünbacher, R. Rabiser. Supporting evolution of product line architectures with variability model fragments. Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), pp. 327 – 330, 2008.

[318] D. Dhungana, P. Grünbacher, R. Rabiser, T. Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. Journal of Systems and Software, pp. 1108 – 1122, 2010.

[319] M. Rosenmüller and N. Siegmund. Automating the configuration of multi software product lines. In Proc. of the Fourth International Workshop on Variability Modelling of Software-intensive Systems (VAMOS'10), pp 123–130, 2010.

[320] S. El-sharkawy, C. Kröher, and K. Schmid. Supporting heterogeneous compositional multi software product lines. MAPLE/SCALE in the Proceedings of the 15th International Software Product Line Conference (SPLC'11), Vol. 2, 2011.

[321] S. El-sharkawy, C. Kröher, and K. Schmid. Support for complex product line population. In the Proceedings of the 15th International Software Product Line Conference (SPLC'11), Vol. 2, 2011.

[322] P. van den Hamer, F. van der Linden, A. Saunders, and H. te Sligte. An integral hierarchy and diversity model for describing product family architectures. In theProceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families, 1998.

[323] Tiziana Margaria, Daniel Meyer, Christian Kubczak, Malte Isberner, and Bernhard Steffen. 2009. Synthesizing Semantic Web Service Compositions with jMosel and Golog. In Proceedings of the 8th International Semantic Web Conference (ISWC '09), Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan (Eds.). Springer-Verlag, Berlin, Heidelberg, 392-407.

[324] Stefan Naujokat, Anna-Lena Lamprecht, Bernhard Steffen, "Tailoring Process Synthesis to Domain Characteristics," Engineering of Complex Computer Systems, IEEE International Conference on, pp. 167-175, 2011 16th IEEE International Conference on Engineering of Complex Computer Systems, 2011.

[325] Petrie, C.; Margaria, T.; Lausen, H.; Zaremba, M. Semantic Web Services Challenge, Springer, 2009.